BUNDESREPUBLIK DEUTSCHLAND

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen: 100 00 423.7

Anmeldetag: 9. Januar 2000

Anmelder/Inhaber: Pact Informationstechnologie GmbH, München/DE

Bezeichnung: Sequenz-Partitionierung auf Zellstrukturen

IPC: G 06 F 15/80

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 7. Dezember 2001
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Holß

A 9161
03/00

PACT13

Aufgabe der Erfindung und Anwendungsbereiche

Die vorliegende Erfindung erstreckt sich auf das Gebiet von programmierbaren und insbesondere während des Betriebes umprogrammierbaren arithmetischen und/oder logischen Bausteinen (VPUs) mit einer Vielzahl von arithmetischen und/oder logischen Einheiten, deren Verschaltung ebenfalls programmierbar und während des Betriebes umprogrammierbar ist. Derartige logische Bausteine sind unter dem Oberbegriff FPGA von verschiedenen Firmen verfügbar. Weiterhin sind mehrere Patente veröffentlicht, die spezielle arithmetische Bausteine mit automatischer Datensynchronisation und verbesserter arithmetischen Datenverarbeitung offenlegen.
Sämtliche beschriebene Bausteine besitzen eine zwei- oder mehrdimensionale Anordnung von logischen und/oder arithmetischen Einheiten (PAEs), die über Bussysteme miteinander verschaltbar sind.
Kennzeichnend für die der Erfindung entsprechenden Bausteine ist, daß sie entweder die nachfolgend aufgelisteten Einheiten besitzen, oder zur erfindungsgemäßen Anwendung diese Einheiten programmiert oder (auch extern) hinzugefügt werden:

1. mindestens eine Einheit (CM) zum Laden der Konfigurationsdaten.
2. PAEs.
3. mindestens ein Interface (IOAG) zu einem oder mehreren Speichern und/oder peripheren Geräten.

Aufgabe der Erfindung ist es, ein Programmierverfahren zur Verfügung zu stellen, das es ermöglicht die beschriebenen Bausteine in gewöhnlichen Hochsprachen effizient zu programmieren und dabei die Vorteile der durch die Vielzahl von Einheiten entstehende Parallelität der beschriebenen

Bausteine weitgehend automatisch, vollständig und effizient zu nutzen.

Stand der Technik

Bausteine der genannten Gattung werden zumeist unter Verwendung gewöhnlicher Datenflusssprachen programmiert. Dabei treten zwei grundlegende Probleme auf:
1. Die Programmierung in Datenflusssprachen ist für Programmierer gewöhnungsbedürftig, tief sequentielle Aufgaben lassen sich nur sehr umständlich beschreiben.
2. Große Applikationen und sequentielle Beschreibungen lassen sich mit den bestehenden Übersetzungsprogrammen (Synthese-Tools) nur bedingt auf die gewünschte Zieltechnologie abbilden (synthetisieren).

Für gewöhnlich werden Applikationen in mehrere Teilapplikationen partitioniert, die dann einzeln auf die Zieltechnologie synthetisiert werden (Fig. 1). Die einzelnen Binärcodes werden dann auf jeweils einen Baustein geladen. Wesentliche Voraussetzung der Erfindung ist das in DE 44 16 881 beschriebene Verfahren, das es ermöglicht, mehrere partitionierte Teilapplikationen innerhalb eines Bausteines zu nutzen, indem die zeitliche Abhängigkeit analysiert wird und über Steuersignale sequentiell die jeweils erforderlichen Teilapplikationen bei einer übergeordneten Ladeeinheit angefordert und von dieser daraufhin auf den Baustein geladen werden.

Existierende Synthese-Tools sind nur bedingt in der Lage Programm-Schleifen auf Bausteine abzubilden (Fig. 2 0201).

Dabei werden **FOR**-Schleifen (0202) als Primitiv-Schleife häufig noch dadurch unterstützt, daß die Schleife vollkommen auf die Ressourcen des Zielbausteines ausgewalzt werden.

WHILE-Schleifen (0203) besitzen im Gegensatz zu FOR-Schleifen keinen konstanten Abbruchswert. Vielmehr wird durch eine Bedingung evaluiert, wann der Schleifenabbruch stattfindet. Daher ist gewöhnlicherweise (wenn die Bedingung nicht konstant ist) zur Synthesezeit nicht bekannt, wann die Schleife abbricht. Durch das dynamische Verhalten können Synthese-Tools diese Schleifen nicht fest auf Hardware abgebildet d.h. auf einen Zielbaustein übertragen werden.

Rekursionen sind grundsätzlich nicht auf Hardware abbildbar, wenn die Rekursionstiefe nicht zur Synthesezeit bekannt und damit konstant ist. Bei der Rekursion werden mit jeder neuen Rekursionsebene neue Ressourcen allokiert. Das würde bedeuten, daß mit jeder Rekursionsebene neue Hardware zur Verfügung gestellt werden muß, was aber dynamisch nicht möglich ist.

Selbst einfache Grundstrukturen sind von Synthesetools nur dann abbildbar, wenn der Zielbaustein ausreichend groß ist, d.h. ausreichende Ressourcen bietet.
Einfache zeitliche Abhängigkeiten (0301) werden durch heutige Synthese-Tools nicht in mehrere Teilapplikationen partitioniert und sind deshalb nur als Ganzes auf einen Zielbaustein übertragbar.
Bedingte Ausführungen (0302) und Schleifen über Bedingungen (0303) sind ebenfalls nur abbildbar, wenn ausreichende Ressourcen auf dem Zielbaustein existieren.

Erfindungsgemäßes Verfahren

Durch das in DE 44 16 881 beschriebene Verfahren ist es möglich Bedingungen zur Laufzeit innerhalb der Hardwarestrukturen der genannten Bausteine zu erkennen und derart dynamisch darauf zu reagieren, daß die Funktion der Hardware entsprechend der eingetretenen Bedingung modifiziert wird, was im wesentlichen durch das Konfigurieren einer neuen Struktur geschieht.

Ein wesentlicher Schritt in dem erfindungsgemäßen Verfahren ist die Partitionierung von Graphen (Applikationen) in zeitlich unabhängige Teilgraphen (Teilapplikationen).

Der Begriff "zeitliche Unabhängigkeit" wird damit definiert, daß die Daten, die zwischen zwei Teilapplikationen übertragen werden durch einen Speicher, gleich welcher Ausgestaltung (also auch mittels einfacher Register), entkoppelt werden. Die ist besonders an den Stellen eines Graphen möglich, an denen eine klare Schnittstelle mit einer begrenzten und möglichst minimalen Menge von Signalen zwischen den beiden Teilgraphen besteht.

Die zeitliche Unabhängigkeit kann in großen Graphen durch das gezielte Einfügen von klar definierten und möglichste einfachen Schnittstellen zum Speichern von Daten in einen Zwischenspeicher herbeigeführt werden (vgl. S_n in Fig. 4). Schleifen weisen grundsätzlich eine starke zeitliche Unabhängigkeit auf, da sie lange Zeit über einer bestimmten Menge von (zumeist) in der Schleife lokalen Variablen arbeiten und nur beim Schleifeneintritt und beim Verlassen der Schleife eine Übertragung der Operanden bzw. des Ergebnisses erfordern.

Durch die zeitliche Unabhängigkeit wird erreicht, daß nach der vollständigen Ausführung einer Teilapplikation die

nachfolgende Teilapplikation geladen werden kann, ohne daß irgendwelche weiteren Abhängigkeiten oder Einflüsse auftreten. Beim Speichern der Daten in den genannten Speicher kann ein Signal (Trigger) generiert werden, das die übergeordneten Ladeeinheit zum Nachladen der nächsten Teilapplikation auffordert. Der Trigger kann bei der Verwendung von einfachen Registern als Speicher immer generiert werden, wenn das Register beschrieben wird. Bei der Verwendung von Speichern, i.b. von solchen die nach dem FIFO-Prinzip arbeiten, ist die Generierung des Triggers von mehreren Bedingungen abhängig. Folgende Bedingungen können beispielsweise einzeln oder kombiniert ein Trigger erzeugen:

- Ergebnis-Speicher voll
- Operanden-Speicher leer
- keine neuen Operanden
- Beliebige Bedingung innerhalb der Teilapplikation, generiert durch z.B.
    * Vergleicher
    * Zähler

Eine Teilapplikation wird im folgenden auch Modul genannt, um die Verständlichkeit aus Sicht der klassischen Programmierung zu erhöhen. Aus demselben Grund werden Signale im folgenden auch Variablen genannt. Dabei unterscheiden sich diese Variablen in einem Punkt wesentlich von herkömmlichen Variablen: Jeder Variable ist ein Statussignal (Ready) zugeordnet, das anzeigt, ob diese Variable einen gültigen Wert besitzt. Wenn ein Signal einen gültigen (berechneten) Wert besitzt, ist das Statussignal Ready; wenn das Signal keinen gültigen Wert besitzt (Berechnung noch nicht abgeschlossen), ist das Statussignal Not_Ready. Das Prinzip ist ausführlich in der Patentanmeldung ~~PACT02~~P196 51 075.9 beschrieben.

Wave-Reconfiguration

Durch eine geeignete Hardwarearchitektur (vgl. Fig. 10/11) ist es möglich mehrere Module zu überlappen. D.h. mehrere Module sind in den PAEs vorkonfiguriert und es kann mit minimalem Zeitaufwand zwischen den Konfigurationen umgeschaltet werden, so daß aus einer Menge von mehreren Konfigurationen pro PAE immer genau eine Konfiguration aktiviert ist.

Wesentlich ist, daß dabei in einer Menge von PAEs in die ein Modul A und B vorkonfiguriert ist, ein Teil der Menge mit einem Teil von A und eine anderer Teil der Menge gleichzeitig mit einem Teil B konfiguriert sein kann. Dabei ist die Trennung der beiden Teile exakt durch die PAE gegeben, in der der Umschaltezustand zwischen A und B auftritt. Das bedeutet, daß ausgehend von einem bestimmten Zeitpunkt bei allen PAEs bei denen vor diesem Zeitpunkt A zur Ausführung aktiviert war B aktiviert ist und bei allen anderen PAEs nach diesem Zeitpunkt immer noch auf A aktiviert ist. Mit steigender Zeit wird bei immer mehr PAEs B aktiviert.

Die Umschaltung erfolgt aufgrund von bestimmten Daten, Zuständen die sich aus der Berechnung der Daten ergeben oder aufgrund beliebiger anderer Ereignisse.

Das bewirkt, daß direkt nach Verarbeitung eines Datenpaketes zu einer anderen Konfiguration umgeschaltet werden kann.

Gleichzeitig/Alternativ kann ein Signal (RECONFIG-TRIGGER) an den CM gesendet werden, das das Vorladen von neuen Konfigurationen durch den CM bewirkt. Das Vorladen kann dabei auf anderen von der aktuellen Datenverarbeitung abhängigen oder unabhängigen PAEs erfolgen. Durch eine Entkopplung der aktiven Konfiguration von den zur Unkonfiguration zur Verfügung stehenden Konfigurationen (vgl. Fig. 10/11) können auch gerade arbeitende (aktive) PAEs, insbesondere auch die PAE, die den RECONFIG-TRIGGER erzeugte, mit neuen

Konfigurationen geladen werden. Dies ermöglicht eine mit der Datenverarbeitung überlappende Konfiguration.

In Figur 13 ist das Grundprinzip der Wave-Reconfiguration (WRC) dargestellt. Dabei wird von einer Reihe von PAEs (PAE1-9) ausgegangen, durch die die Daten pipelineähnlich laufen. Es wird ausdrücklich darauf hingewiesen, daß WRC nicht auf Pipelines beschränkt ist und die Vernetzung und Gruppierung der PAEs jede beliebige Form annehmen kann. Die Darstellung wurde jedoch gewählt um ein einfaches Beispiel zum besseren Verständnis zu zeigen.
In Fig. 13a läuft ein Datenpaket in die PAE1. Die PAE besitzt 4 mögliche Konfigurationen (A, F, H, C), die durch eine geeignete Hardware (vgl. Fig. 10/11) wählbar sind. Die Konfiguration F ist in in PAE1 für das aktuelle Datenpaket aktiviert (schraffiert dargestellt).
Im nächsten Takt läuft das Datenpakt nach PAE2 und ein neues Datenpaket erscheint in PAE1. Auch in PAE2 ist F aktiv. Zusammen mit dem Datenpaket erscheint ein Ereignis (↑1) bei PAE1. Das Ergnis entsteht durch Eintreffen eines beliebigen Ereignisses von aussen bei der PAE (z.B. eines Statusflags oder Triggers) oder wird innerhalb der PAE durch die ausgeführte Berechnung generiert.
In Fig. 13c wird in PAE1 aufgrund des Ereignisses (↑1) die Konfiguration H aktiviert, gleichzeitig erscheint ein neues Ereignis (↑2), das im nächsten Takt (Fig. 13d) die Aktivierung von Konfiguration A bewirkt.
In Fig. 13e trifft (↑3) bei PAE1, die das Überschreiben von F mit G bewirkt (Fig. 13f). Durch das Eintreffen von (↑4) wird G aktiviert (Fig. 13g). (↑5) bewirkt das Laden von K anstelle von C (Fig 13h,i) und (↑6) lädt und startet F anstelle von H (Fig. 13j).

In Fig. 13 bewegt eine Welle von Umkonfigurationen aufgrund von Ereignissen durch eine Menge von PAEs, die 2- oder mehrdimensional ausgestaltet sein kann.
Es ist nicht zwingend notwendig, daß eine einmal stattfindende Umkonfiguration durch die gesamten Fluß hinweg stattfindet. Beispielsweise könnte die Umkonfiguration mit der Aktivierung von A aufgrund des Ereignisses (↑2) nur lokal in den PAEs1 bis 3 und PAE7 stattfinden, während in allen anderen PAEs weiterhin die Konfiguration H aktiviert bleibt.
Mit anderen Worten:

a) Es ist möglich, daß ein Ereigniss nur lokal auftritt und daher nur lokal eine Umaktivierung zur Folge hat,
b) ein globales Ereignis, hat möglicherweise keine Auswirkung auf manche PAEs; abhängig vom ausgeführten Algorithmus.

Bei den PAEs die nach (↑2) weiterhin H aktiviert halten, kann selbstverständlich das Eintreffen des Ereignisses (↑3) vollkommen andere Auswirkungen haben, (I) wie etwa das Aktivieren von C statt dem Laden von G, (ii) andererseits könnte (↑3) auf diese PAEs auch gar keinen Einfluß haben.

Das Prozessormodell

Die in den folgenden Figuren gezeigten Graphen besitzen als Graphenknoten immer ein Modul, wobei davon ausgegangen wird, daß mehrere Module auf einen Zielbaustein abgebildet werden können. Das heißt, obwohl alle Module zeitlich voneinander unabhängig sind, wird nur bei ~~nach~~ den Modulen eine Umkonfiguration durchgeführt, ~~bzw.~~ und/oder ein Datenspeicher eingefügt, die mit einem vertikalen Strich und Δt markiert sind. Dieser Punkt wird Umkonfigurationszeitpunkt genannt.

Der Umkonfigurationszeitpunkt ist abhängig von den bestimmten Daten oder den Zuständen die sich aus der Verarbeitung der bestimmten Daten ergeben.

Das bedeutet zusammenfassend:

1. Große Module können an geeigneten Stellen partitioniert werden und in kleine zeitlich voneinander unabhängige Module zerlegt werden.
2. Bei kleinen Modulen die gemeinsam auf einen Zielbaustein abgebildet werden können, wird auf die zeitliche Unabhängigkeit verzichtet. Dadurch werden Konfigurationsschritte eingespart und die Datenverarbeitung beschleunigt.
3. Die Umkonfigurationszeitpunkte werden entsprechend der Ressourcen der Zielbausteine positioniert. Dadurch ist eine beliebige Skalierung der Graphenlänge gegeben.
4. Module können überlagert konfiguriert werden.
5. Die Umkonfiguration von Modulen wird durch die Daten selbst oder dem Ergebnis der Verarbeitung der Daten gesteuert.

~~Copy-And-Compute~~ Virtual Machine Modell

Die Grundlagen der Datenverarbeitung mit VPU-Bausteinen sind entsprechend der vorhergehenden Abschnitte hauptsächlich datenflußorientiert. Um sequentielle Programme mit ordentlicher Leistung abzuarbeiten, ist es jedoch notwendig ein sequentielles Datenverarbeitungsmodell zur Verfügung zu haben. Dabei sind oftmals die Sequenzer in den einzelnen PAEs nicht ausreichend.

Die Architektur von VPUs ermöglicht jedoch grundsätzlich den Aufbau von beliebig komplexen Sequenzern aus einzelnen PAEs.

Das bedeuted:

1. Es können komplexe Sequenzer konfiguriert werden, die exakt den Anforderungen des Algorithmus entsprechen.

2. Der Datenfluß kann, wie bereits ausführlich diskutiert, exakt die Rechenschritte des Algorithmus repräsentieren.

Dadurch kann eine Virtuelle Maschine auf VPUs implementiert werden, die insbesondere auch den sequentiellen Anforderungen eines Algorithmus entspricht.

Hauptvorteil der VPU-Architektur ist, daß ein Algorithmus durch einen Compiler so zerteilt werden kann, daß die Datenflußteile extrahiert werden durch einen "optimalen" Datenfluß repräsentiert werden, indem ein angepaßter Datenfluß konfiguriert wird UND die sequentiellen Teile des Algorithmus durch einen "optimalen" Sequenzer repräsentiert werden, indem ein angepaßter Sequenzer konfiguriert wird. Dabei können gleichzeitig mehrere Sequenzer und Datenflüsse auf einer VPU untergebracht werden, ausschließlich abhängig von den zur Verfügung stehenden Ressourcen.

Durch die große Anzahl an PAEs entstehen im Betrieb innerhalb einer VPU sehr viele lokalen Zustände. Bei Taskwechseln oder Unterprogramm-Aufrufen (Interrupts) müssen diese Zustände gesichert werden (vgl. PUSH/POP bei Standardprozessoren). Dies ist jedoch aufgrund der Menge an Zuständen nicht sinnvoll möglich.

Um die Zustände auf eine handhabbare Menge zu reduzieren muß zwischen zwei Arten von Zuständen unterschieden werden:

1. Zustandsinformationen des Maschinenmodells (MACHINE-STATE). Diese Zustandsinformationen sind nur innerhalb der Abarbeitung eines bestimmten Modules gültig und werden auch nur lokal in den Sequenzern und Datenflußeinheiten dieses bestimmten Modules verwendet. D.h. diese MACHINE-STATEs repräsentieren die Zustände, die in Prozessoren nach dem

Stand der Technik verdeckt innerhalb der Hardware ablaufen, implizit in den Befehlen und den Verarbeitungsschritten sind und nach Ablauf eines Befehles keine weitere Information für nachfolgende Befehle beinhalten. Derartige Zustände brauchen nicht gesichert zu werden. Bedingung dafür ist, daß Interrupts nur nach kompletter Ausführung aller gerade aktiven Module durchgeführt werden. Stehen Interrupts zur Ausführung an, werden keine neuen Module geladen, sondern nur noch aktive abgearbeitet; ebenfalls werden den aktiven Modulen, soweit es der Algorithmus zuläßt keine neuen Operanden mehr zugeführt. Dadurch wird ein Modul zu einer atomaren nicht unterbrechbaren Einheit, vergleichbar mit einer Instruktion eines Prozessors nach dem Stand der Technik.

2. Zustände der Datenverarbeitung (DATA-STATE). Die datenbezogenen Zustände müssen beim Auftreten eines Interrupts entsprechend den Prozessormodellen nach dem Stand der Technik gesichert und in den Speicher geschrieben werden. Das sind bestimmte notwendige Register und Flags oder - entsprechend der Begriffe der VPU-Technologie - Trigger.

Bei den DATA-STATEs kann die Handhabung je nach Algorithmus weiter vereinfacht werden. Zwei grundlegende Strategien werden im Folgenden nähers erläutert:

1. Mitlaufen der Zustandsinformation

Dabei werden alle relevanten und zu einem späteren Zeitpunkt benötigten Zustandsinformationen von einem Modul zum nächsten übertragen, wie es in Pipelines oftmals standardmäßig implementiert ist. Die Zustandsinformationen werden dann zusammen mit den Daten implizit in einem Speicher abgelegt, sodaß die Zustände bei einem Abruf der Daten zugleich zur Verfügung stehen. Ein explizites Handhaben der

Zustandsinformationen i.b. mittels PUSH und POP entfällt dadurch, was je nach Algorithmus einerseits zu einer wesentlichen Beschleunigung der Abarbeitung und andererseits zu einer vereinfachten Programmierung führt.
Die Zustandsinformaton kann wahlweise entweder mit dem jeweiligen Datenpaket gespeichert werden, oder nur im Falle eines Interrupts gesichert und besonders gekennzeichnet werden.

2. Sichern der Reentry Adresse
Bei der Verarbeitung von großen Datenmengen, die in einem Speicher abgelegt sind, ~~ist~~ kann es sinnvoll sein, die Adresse mindestens einer der Operanden des gerade verarbeiteten Datenpaketes mit dem Datenpaket zusammen durch die PAEs zu leiten. Dabei wird die Adresse nicht modifiziert, sondern steht beim Schreiben des Datenpaketes in ein RAM als Pointer auf den letzten verarbeiteten Operanden zur Verfügung.
Dieser Pointer kann wahlweise entweder mit dem jeweiligen Datenpaket gespeichert werden, oder nur im Falle eines Interrupts gesichert und besonders gekennzeichnet werden.
Insbesondere, wenn sämtliche Pointer auf die Operanden durch eine Adresse (oder eine Gruppe von Adressen) berechnet werden können ist es sinnvoll nur eine Adresse (oder eine Gruppe von Adressen) zu sichern.

"ULIW"-"UCISC"-Modell
FÜr das Verständnis dieses (einem Prozessor nach dem Stand der Technik sehr ähnlichen) Modells ist eine Erweiterung der Betrachtungsweise der Architektur von VPUs erforderlich. Dabei dient das Virtual-Machine Modell als Grundlage.
Das Array aus PAEs (PA) wird als in ihrer Architektur konfigurierbare Recheneinheit betrachtet. Der/die CM(s)

stellen eine Ladeeinheit (LOAD-UNIT) für Opcodes dar. Die IOAG(s) übernehmen das Businterface und/oder den Registersatz.

Diese Anordnung ermöglicht zwei grundsätzliche Funktionsweisen, die im Betrieb gemischt verwendbar sind:

1. Eine Gruppe von PAEs (das kann auch eine PAE sein) wird zur Ausführung eines komplexen Befehls oder Befehlsfolge konfiguriert und danach werden die auf diesen Befehl bezogenen Daten (das kann auch ein einziges Datenwort sein) verarbeitet. Danach wird diese Gruppe umkonfiguriert, zur Abarbeitung des nächsten Befehles. Dabei kann sich die Größe und Anordnung der Gruppe ändern. Gemäß den bereits besprochenen Partitionierungstechnologien obliegt es dem Compiler, möglichst optimale Gruppen zu schaffen. Durch den CM werden Gruppen als Befehle auf den Baustein "geladen", dadurch ist des Verfahren mit dem bekannten VLIW vergleichbar, nur daß erheblich mehr Rechenwerke verwaltet werden UND die Vernetzungsstruktur zwischen den Rechenwerken ebenfalls vom Instruktionswort abgedeckt werden kann (Ultra Large Instruction Word = "ULIW"). Ein Instruktionswort entspricht dabei einem Modul. Mehrere Module können gleichzeitig verarbeitet werden, sofern es die Abhängigkeit der Daten zuläßt und genügend Ressourcen auf dem Baustein frei sind. Wie bei VLIW-Befehlen wird für gewöhnlich nach Ausführen des Instruktionswortes sofort das nächste Instruktionswort geladen. Zur zeitlichen Optimierung ist es dabei möglich das nächste Instruktionswort bereits während der Ausführung vorzuladen (vgl. Fig. 10). Bei mehreren möglichen nächsten Instruktionswörten können mehrere vorgeladen werden und vor der Ausführung wird z.B. durch ein Triggersignal das korrekte Instruktionswort ausgewählt. BEISPIEL MIT IF..ELSE ZEICHNEN!

2. Eine Gruppe von PAEs (das kann auch eine PAE sein) wird zur Ausführung einer häufig gebrauchten Befehlsfolge konfiguriert. Die Daten, das kann auch hier ein einzelnes Datenwort sein, werden bei Bedarf der Gruppe zugeführt und von der Gruppe empfangen. Diese Gruppe bleibt über eine Vielzahl von Takten ohne Umkonfiguration bestehen. Vergleichbar ist diese Anordnung mit einem speziellen Rechenwerk in einem Prozessor nach dem Stand der Technik (z.B. MMX), das für Spezialaufgaben vorgesehen ist und nur bei Bedarf verwendet wird. Durch diesen Ansatz sind Spezialbefehle entsprechend des CISC[x]-Prinzipes generierbar, mit dem Vorteil, daß diese Befehle anwendungsspezifisch geschaffen werden können (Ultra-CISC = "UCISC").

P 19651075.9

Erweiterung des RDY/ACK-Protokolls (vgl. ~~PACT02~~)

In ~~PACT02~~ P19651075.5 ist ein RDY/ACK-Standardprotokoll beschrieben, das die wesentlichen Anforderungen gemäß den Synchronisationen von DE 44 16 881 in Hinblick auf eine typische Datenflußapplikation beschreibt. Nachteil des Protokolles ist, daß lediglich Daten gesendet und der Empfang bestätigt werden kann. Der umgekehrte Fall, indem Daten angefordert werden und das Versenden bestätigt wird (im Folgenden REQ/ACK genannt, ist zwar elektrisch mit demselben Zweidrahtprotokoll lösbar, jedoch semantisch nicht erfaßt. Das gilt insbesondere, wenn REQ/ACK und RDY/ACK gemischt betrieben werden.
Daher wird die klare Unterscheidung der Protokolle eingeführt:
RDY: Daten liegen beim Versender für den Empfänger bereit
REQ: Daten werden vom Empfänger beim Versender angefordert
ACK: Allgemeine Bestätigung für erfolgten Emfang oder Versand

[x] CISC = complex instruction set computers

09.01.00

(Prinzipiell könnten auch zwischen ACK für ein RDY und einem ACK für ein REQ unterschieden werden, jedoch ist in den Protokollen die Semantik des ACKs für gewöhnlich implizit).

Speichermodell

In VPUs können Speicher integriert werden (einer oder mehrere), die ähnlich einer PAE angesprochen werden. Es wird im folgenden ein Speichermodell beschrieben, das gleichzeitig ein Interface zu externer Peripherie und/oder externem Speicher darstellt:

Ein VPU-interner Speicher mit PAE-ähnlichen Busfunktionen kann verschiedene Speichermodi darstellen:

1. Standardspeicher
2. Lookup-Tabelle
3. FIFO
4. LIFO

Dem Speicher ist ein steuerbares Interface zugeordnet, das Speicherbereiche wahlweise wort- oder blockweise schreibt oder liest.

Dadurch ergeben sich folgende Nutzungsmöglichkeiten:

1. Entkopplung von Datenströmen (FIFO)
2. Schneller Zugriff auf selektierte Speicherbereiche eines externen Speichers, was eine Cacheähnliche Funktion darstellt (Standardspeicher, Lookup-Tabelle)
3. Stack mit variierbarer Tiefe (LIFO)

Dabei kann das Interface benutzt werden, es ist jedoch nicht zwingend notwendig, wenn die Daten z.B. ausschließlich lokal in der VPU verwendet werden und der Speicherplatz eines internen Speichers ausreicht.

09.01.00

Stack Modell

Durch Verwendung des REQ/ACK-Protokolls und der internen Speicher im LIFO-Modus kann ein einfacher Stack-Prozessor aufgebaut werden. Dabei werden temporäre Daten von den PAEs auf den Stack geschrieben und bei Bedarf von dem Stack geladen. Die hierfür notwendigen Compilertechnologien sind hinreichend bekannt.
Durch die variierbare Stacktiefe, die durch einen Datenaustausch des internen Speicher mit einem externen Speicher erreicht wird, kann der Stack beliebig groß werden.

Akkumulator Modell

Jede PAE kann eine Recheneinheit nach dem Akkumulatorprinzip darstellen. Wie aus ~~PACT02~~ P19651076.9 bekannt ist, es möglich, die Ausgangsregister auf den Eingang der PAE rückzukoppeln. Dadurch entsteht ein Akkumulator nach dem Stand der Technik. In Verbindung mit dem Sequenzer nach Fig. 11 lassen sich einfache Akkumulator-Prozessoren aufbauen.

Register Modell

Durch Verwendung des REQ/ACK-Protokolls und der internen Speicher im Standardspeicher-Modus kann ein einfacher Register-Prozessor aufgebaut werden. Dabei werden die Registeradressen von einer Gruppe von PAEs generiert, während eine andere Gruppe von PAEs die Verarbeitung der Daten übernimmt.

Die verwendeten Maschinenmodelle können innerhalb einer VPU beliebig kombiniert werden. Auch innerhalb eines Algorithmus

09.01.00

kann, je nach dem, welches Modell jeweils optimal ist, zwischen den Modellen gewechselt werden.

Wird einem Register-Prozessor ein weiterer Speicher zugefügt, von dem die Operanden gelesen werden und in den die Ergebnisse geschrieben werden, kann eine Load/Store-Prozessor aufgebaut werden. Dabei können mehrere verschiedene Speicher zugeordnet werden, indem die einzelnen Operanden und das Ergebnis getrennt behandelt wird.
Diese Speicher arbeiten dann quasi als Load/Store-Einheit und stellen eine Art Cache für den externen Speicher dar. Die Adressen werden durch von der Datenverarbeitung separierte PAEs berechnet.

Pointer Reordering
Hochsprachen wie C/C++ verwenden häufig Pointer, die sehr schlecht durch Pipelines gehandhabt werden können. Wenn ein Pointer erst direkt vor dem Verwenden der Datenstrukturen auf die er zeigt, berechnet wird, kann häufig die Pipeline nicht schnell genug gefüllt werden und die Verarbeitung wird speziell in VPUs ineffizient.
Sicherlich ist es sinnvoll, bei der Programmierung von VPUs möglichst keine Pointer zu verwenden, jedoch ist das oftmals nicht möglich.
Eine Lösung hierfür ist, die Pointerstrukturen durch den Compiler so umzusortieren, daß die Pointeradressen möglichst lange vor deren Verwendung berechnet werden. Gleichzeitig sollte es möglichst wenig direkte Abhängigkeiten zwischen einem Pointer und den Daten, auf die er zeigt, geben.

Figuren

In Fig. 4a sind einige grundlegenden Eigenschaften des erfindungsgemäßen Verfahrens dargestellt:
Die Module des Types A sind zu einer Gruppe zusammengefaßt und besitzen am Ende einen bedingten Sprung, entweder nach B1 oder B2. An dieser Position (0401) ist ein Umkonfigurationspunkt eingefügt, da es sinnvoll ist die Zweige des bedingten Sprunges als jeweils eine Gruppe zu betrachten (Fall 1). Würden dagegen beide Zweige von B (B1 und B2) zusätzlich zusammen mit A auf den Zielbaustein passen (Fall 2), wäre es sinnvoll, nur einen Umkonfigurationspunkt bei 0402 einzufügen, da dadurch die Zahl der Konfigurationen verringert wird und sich die Verarbeitungsgeschwindigkeit erhöht. Beide Zweige (B1 und B2) springen bei 0402 nach C.

Die Konfiguration der Zellen auf dem Zielbaustein ist in Fig. 4b schematisch dargestellt. Dabei werden die Funktionen der einzelnen Graphenknoten auf die Zellen des Zielbausteins abgebildet. Jeweils eine Zeile stellt eine Konfiguration dar. Die gestrichelten Pfeile bei einem Zeilenwechsel zeigen eine Umkonfiguration an. S_n ist eine datenspeichernde Zelle, von beliebiger Ausgestaltung (Register, Speicher, etc.). Dabei ist S_nI ein Speicher, der Daten entgegennimmt und S_nO ein Speicher, der Daten ausgibt. Der Speicher S_n ist für gleiche n jeweils derselbe, I und O kennzeichnen die Datentransferrichtung.

Beide Fälle des bedingten Sprunges (Fall 1, Fall 2) sind dargestellt.

Das Modell in Fig. 4 entspricht einem Datenflußmodell, jedoch mit der wesentlichen Erweiterung des Umkonfigurationspunkts und der damit erreichbaren Partitionierung des Graphen, wobei

die zwischen den Partitionen übertragenen Daten zwischengespeichert werden.

Im Modell von Fig. 5a wird aus einer beliebigen Graphenmenge und -Konstellation (0501) selektiv ein Graph B_n aus einer Menge von Graphen B aufgerufen. Nach der Ausführung von B gelangen die Daten nach 0501 zurück.
Wird in 0501 ein ausreichend großer Sequencer (A) implementiert, ist mit dem Modell ein den typischen Prozessoren sehr ähnliches Prinzip implementierbar. Dabei gelangen
1. Daten in den Sequenzer A, die dieser als Befehle dekodiert und entsprechend dem "von Neumann"-Prinzip darauf reagiert;
2. Daten in den Sequenzer A, die als Daten betrachtet werden und an ein fest konfiguriertes Rechenwerk C zur Berechnung weitergeleitet werden.

Der Graph B stellt selektierbar ein besonderes Rechenwerke und/oder besondere Opcodes für bestimmte Funktionen zur Verfügung und wird alternativ zur Beschleunigung von C verwendet. Beispielsweise kann B1 ein optimierter Algorithmus zu Berechnung von Matrixmultiplikationen sein, während B2 einen FIR-Filter und B3 eine Mustererkennung darstellt. Entsprechend eines Opcodes der von 0501 dekodiert wird, wird der geeignete bzw. entsprechende Graph B aufgerufen.

Fig. 5b schematisiert die Abbildung auf die einzelnen Zellen, wobei in 0502 der pipelineartige Rechenwerks-Character symbolisiert wird.

Während in den Umkonfigurationspunkten von Fig. 4 vorzugsweise größere Speicher zum Zwischenspeichern der Daten eingefügt werden, ist eine einfache Synchronisation der Daten in den

Umkonfigurationspunkten von Fig. 5 ausreichend, da der Datenstrom vorzugsweise als ganzer durch den Graphen B läuft und der Graph B nicht weiter partitioniert ist; dadurch ist das Zwischenspeichern der Daten überflüssig.

In Fig. 6a sind verschiedene Schleifen dargestellt. Schleifen können grundsätzlich auf drei Arten behandelt werden:
1. Hardware-Ansatz: Schleifen werden vollständig ausgewalzt auf die Zielhardware abgebildet (0601a/b). Wie bereits erläutert ist dies nur bei wenigen Schleifenarten möglich.
2. Datenfluß-Ansatz: Innerhalb des Datenflußes werden Schleifen über mehrere Zellen hinweg aufgebaut (0602a/b). Das Ende der Schleife wird auf den Schleifenanfang rückgekoppelt.
3. Sequenzer-Ansatz: Ein Sequenzer mit minimalem Befehlssatz führt die Schleife aus (0603a/b). Dabei sind die Zellen der Zielbausteine so ausgestaltet, daß sie den entsprechenden Sequenzer beeinhalten (vgl. Fig. 11a/b).

Durch eine geeignete Zerlegung von Schleifen kann deren Ausführung ggf. optimiert werden:
1.Unter Verwendung von Optimierungsmethoden nach dem Stand der Technik läßt sich häufig der Schleifenrumpf, also der wiederholt auszuführende Teil, dadurch optimieren, daß bestimmte Operationen aus der Schleife entfernt werden und vor oder hinter die Schleife gestellt werden (0604a/b). Dadurch wird die Menge der zu sequencenden Befehle erheblich reduziert. Die entfernten Operationen werden nur einmal vor bzw. nach Ausführung der Schleife durchlaufen.
2. Eine weitere Optimierungsmöglichkeit ist das Teilen von Schleifen in mehrere kleinere oder kürzere Schleifen. Dabei findet die Teilung derart statt, daß mehrere parallele oder mehrere sequentielle (0605a/b) Schleifen entstehen.

Fig. 7 verdeutlicht die Implementierung einer Rekursion. Dabei werden dieselben Ressourcen (0701) in Form von Zellen für jede Rekursionsebene (1-3) verwendet. Die Ergebnisse einer jeden Rekursionsebene (1-3) werden beim Aufbau (0711:÷) in einen nach dem Stack-Prinzip aufgebauten Speicher (0702) geschrieben. Gleichzeitig mit dem Abbau (0712:) der Ebenen wird der Stack abgebaut.

In Fig. 14 wir das Virtual-Machine-Modell dargestellt. Aus einem externen Speicher werden Daten (1401) und zu den Daten gehörende Zustände (1402) in eine VPU (1403) gelesen. 1401/1402 werden über eine von der VPU generierte Adresse 1404 selektiert. Innerhalb der VPU sind PAEs zu unterschiedlichen Gruppen zusammengefaßt (1405, 1406, 1407). Jede Gruppe besitzt einen datenverarbeitenden Teil (1408), der lokale impiflzite Zustände (1409) besitzt, die keinen Einfluß auf die umliegenden Gruppen besitzt. Daher werden dessen Zustände nicht außerhalb der Gruppe weitergeleitet. Er kann jedoch von den externen Zuständen abhängig sein. Ein weiterer Teil (1410) generiert Zustände, die Einfluß auf die umliegenden Gruppen haben.
Die Daten und Zustände der Ergebnisse werden in einen weiteren Speicher (1411, 1412) abgelegt. Gleichzeitig kann die Adresse von Operanden (14004) als Pointer gespeichert (1413) werden. Zur zeitliche Synchronisation kann 1404 über Register (1414) geführt werden.

In Fig. 14 ist zur Verdeutlichung ein einfaches Modell dargestellt. Die Vernetzung und Gruppierung kann wesentlich komplexer sein als in diesem Modell. Ebenfalls können Zustände und Daten auch an weitere Module als den Nachfolgenden übertragen werden. Es ist möglich, daß Daten an andere Module übertragen werden als die Zustände. Sowohl Daten als auch

09.01.00

Zustände eines bestimmten Moduls können von mehreren unterschiedlichen Modulen empfangen werden. Innerhalb einer Gruppe kann 1408, 1409 und 1410 vorhanden sein, Abhängig vom Algorithmus können auch einzelne Teile fehlen (z.B. 1410 und 1409 vorhanden, 1410 jedoch nicht).

In Figur 15 ist dargestellt, wie aus einem Verarbeitungsgraphen Teilapplikationen extrahiert werden. Dabei wird der Graph so zerlegt, daß lange Graphen sinnvoll zerteilt werden und in Teilapplikationen (A,B) abgebildet werden. (!!!A NOCH LÄNGER MACHEN!!!) Nach Sprüngen werden neue Teilgraphen gebildet (C,K) wobei für jeden Sprung ein getrennter Teilgraph gebildet wird.
Jeder Teilgraph ist in dem ULIW-Modell von der CM (vgl. PACT10, dh, DE 198 078 72 u. deren Patentfamilie) getrennt ladbar. Wesentlich ist, daß Teilgraphen durch die Mechanismen in PACT10 DE 198 08 872.2, WO 99/44147 usw verwaltet werden können. Dazu gehört insbesondere das intelligente Vorladen, Laden und Löschen der Teilapplikationen.
Das ULIW-Modell unterscheidet sich im Wesentlichen vom VLIW-Modell, indem es
1. Das Routing der Daten mit beinhaltet
2. Größere Instruktionswörter bildet.

Ebenfalls kann das beschriebene Verfahren der Partitionierung von Compilern für heutige Standardprozessoren nach dem RISC/CISC-Prinzip ebenso eingesetzt werden. Wird dann eine Einheit (CT) nach PACT10 zur Steuerung des Befehls-Caches verwendet, kann dieser erheblich optimiert und beschleunigt werden.
Dazu werden "normale" Programme entsprechend in Teilapplikationen partitioniert. Gemäß PACT10 werden Verweise auf mögliche nachfolgende Teilapplikationen eingeführt (1501, 1502). Dadurch kann eine CT die Teilapplikationen bereits in

< dh, ein daten verarbeitender Teil (1408), der bzw. implizite Zustände (1409) benötigt, und einen weiteren Teil 1410, der Zustände generiert, die Einfluß auf die umliegenden Gruppen hinten.

den Cache vorladen bevor sie benötigt werden.Bei Sprüngen wird nur die angesprungen Teilapplikation ausgeführt, die andere(n) werden später durch neue Teilapplikationen überschrieben. Neben dem intelligenten Vorladen hat das Verfahren den weiteren Vorteil, daß die Größe der Teilapplikationen beim Laden bereits bekannt ist. Dadurch können optimale Bursts beim Zugriff auf die Speicher von der CT ausgeführt werden, was den Speicherzugriff wiederum erheblich beschleunigt.

Figur 16 zeigt den Aufbau eines Stack-Prozessors. Durch das PAE-Array (1601) werden Protokolle generiert um auf einen als LIFO konfigurierten Speicher (1602) zu schreiben oder zu lesen. Dabei wird ein RDY/ACK-Protokoll zum Schreiben und REQ/ACK-Protokoll zum Lesen verwendet. Die Vernetzung und Betriebsmodi werden von der CT (1603) konfiguriert. 1602 kann dabei seinen Inhalt auf den externen Speicher (1604) auslagern.

Eine Reihe der PAEs sollen in diesem Beispiel als Register-Prozessor arbeiten (Figur 17). Jede PAE besteht aus einem Rechenwerk (1701) und einem Akkumulator (1702) auf den das Ergebnis von 1701 rückgekoppelt (1703) ist. Damit stellt in diesem Beispiel jede PAE einen Akkumulator-Prozessor dar. Eine PAE (1705) liest und schreibt die Daten in den als Standardspeicher konfigurierten RAM (1704). Eine weitere PAE (1706) generiert die Registeradressen.
Oftmals ist es sinnvoll eine getrennt PAE zum Lesen der Daten zu verwenden. Dann würde 1705 nur schreiben und die PAE 1707 lesen. Dabei wird eine weitere PAE (1708, gestrichelt unterlegt) zum Generieren der Leseadressen einzuführen.
Es ist nicht zwingend notwendig getrennte PAEs zum Generieren der Adressen zu verwendet. Oftmals sind die Register implizit

und können dann als Konstanten konfiguriert werden von den datenverarbeitenden PAEs gesendet werden.
Die Verwendung von Akkumulator-Prozessoren für einen Register-Prozessor ist beispielshaft. Ebenso können zum Aufbau von Registerprozessoren PAEs ohne Akkumulator verwendet werden.
Die in Figur 17 gezeigte Architektur kann zur Ansteuerung von Registern als auch zum Ansteuern einer Load/Store-Einheit dienen.
Bei der Verwendung als Load/Store-Einheit ist es fast zwingend notwendig, einen externen RAM (1709) nachzuschalten, sodaß 1704 nur einen temporären Ausschnitt aus 1709, quasi als Cache, darstellt.
Auch bei der Verwendung von 1704 als Register-Bank ist es teilweise sinnvoll einen externen Speicher nachzuschalten. Dadurch können PUSH/POP Operationen nach dem Stand der Technik durchgeführt werden, die den Registerinhalt in einen Speicher schreiben oder aus diesem Lesen.

In Figur 18 ist als Beispiel eine komplexe Maschine abgebildet bei der das PAE-Array (1801) einerseits einen Load/Store-Einheit (1802) mit nachgeschaltetem RAM (1803) ansteuert und gleichzeitig eine Register-Bank (1804) mit nachgeschaltetem RAM (1805) aufweist. 1802 und 1804 können jeweils von einer PAE oder einer beliebigen Gruppe von PAEs angesteuert werden. Die Einheit wird gemäß dem VPU-Prinzip von einer CT (1806) gesteuert.
Wichtig ist, daß zwischen der Load/Store-Einheit (1802) und der Register-Bank (1804) und deren Ansteuerung kein wesentlicher Unterschied besteht.

In Figur 19 ist ein Beispiel für das Umsortieren von Graphen gezeigt.

~~Hochsprachenbeispiele~~Programmierbeispiele

Ein Modul kann beispielsweise folgendermaßen deklariert werden:

```
module example1
input  (var1, var2 : ty1; var3 : ty2).
output (res1, res2 : ty3).
begin
...
register <regname1> (res1).
register <regname2> (res2).
terminate@ (res1 & res2; 1).
end.
```

module kennzeichnet den Beginn eines Modules.
input/output definiert die Ein-/Ausgangsvariablen mit den Typen ty_n.
begin ... **end** markieren den Rumpf des Modules.
register <regname1/2> übergibt das Ergebnis an den Output, wobei des Ergebnis in dem durch <regname1/2> spezifizierten Register zwischengespeichert wird. <regname1/2> ist dabei eine globale Referenz auf ein bestimmtes Register.

Als weitere Übergabemodi an den Output stehen beispielsweise folgende Speicherarten zur Verfügung:
fifo <fifoname>, wobei die Daten an einen nach dem FIFO-Prinzip arbeitenden Speicher übergeben werden. <fifoname> ist dabei eine globale Referenz auf einen bestimmten, im FIFO-Modus arbeitenden Speicher. **terminate@** wird dabei um den Parameter bzw. das Signal "fifofull" erweitert, der/das anzeigt, daß der Speicher voll ist.

stack <stackname>, wobei die Daten an einen nach dem Stack-Prinzip arbeitenden Speicher übergeben werden. <stackname> ist dabei eine globale Referenz auf einen bestimmten, im Stack-Modus arbeitenden Speicher.

terminate@ unterscheidet die Programmierung entsprechend des erfindungsgemäßen Verfahrens von der herkömmlichen sequentiellen Programmierung. Der Befehl definiert das Abbruchkriterium des Modules. Die Ergebnisvariablen res1 und res2 werden von terminate@ nicht mit ihrem tatsächlichen Wert evaluiert, statt dessen wird nur die Gültigkeit der Variablen (also deren Statussignal) geprüft. Dazu werden die beiden Signale res1 und res2 boolsch miteinander verknüpft, z.B. durch eine UND-, ODER- oder XOR-Operation. Sind beide Variablen gültig, terminiert das Modul mit dem Wert 1. Das bedeutet, ein Signal mit dem Wert 1 wird an die übergeordneten Ladeeinheit weitergeleitet, woraufhin die übergeordneten Ladeeinheit das nachfolgende Module lädt.

```
module example2
input  (var1, var2 : ty3; var3 : ty2).
output (res1 : ty4).
begin
register <regname1> (var1, var2).
...
fifo <fifoname1> (res1, 256).
terminate@ (fifofull(<fifoname1>); 1).
end.
```

register wird in diesem Beispiel über input-Daten definiert. Dabei ist <regname1> derselbe wie in example1. Dies bewirkt,

daß das Register, das die output-Daten in example1 aufnimmt, die input-Daten für example2 zur Verfügung stellt.

fifo definiert einen FIFO-Speicher der Tiefe 256 für die Ausgangsdaten res1. Das Full-Flag (fifofull) des FIFO-Speichers wird in **terminate@** als Abbruchkriterium verwendet.

```
module main
input  (in1, in2 : ty1; in3 : ty2).
output (out1 : ty4).
begin
define <regname1> : register(234).
define <regname2> : register(26).
define <fifoname1> : fifo(256,4).  // FIFO Tiefe 256
...
(var12, var72) = call example1 (in1, in2, in3).
...

(out1) = call example2 (var12, var72, var243).
...
signal (out1).
terminate@ (example2).
end.
```

define definiert eine Schnittstelle für Daten (Register, Speicher, etc). Bei der Definition werden die erforderlichen Ressourcen, sowie die Bezeichnung der Schnittstelle angegeben. Da die Ressourcen jeweils nur einmal zur Verfügung stehen, müssen sie eindeutig angegeben werden. Damit ist die Definition global, d.h. die Bezeichnung gilt für das gesamte Programm.

call ruft ein Modul als Unterprogramm auf.

signal definiert ein Signal als Ausgangssignal, ohne daß eine Zwischenspeicherung verwendet wird.

Durch **terminate@** (example2) wird das Modul main terminiert, sobald das Unterprogramm example2 terminiert.

Durch die globale Deklaration "define ..." ist es prinzipiell nicht mehr notwendig, die so definierten input/output Signale in die Schnittstellen-Deklaration der Module aufzunehmen.

Die Zustandsinformationen des Prozessormodells

Zur Bestimmung der Zustände innerhalb eines Graphen werden die Statusregister der einzelnen Zellen (PAEs) über ein zusätzlich zum Datenbus (0801) existierendes, frei rout- und segmentierbares Status-Bussystem (0802) allen anderen Rechenwerken zur Verfügung gestellt (Fig. 8b). Das bedeutet, daß eine Zelle (PAE X) die Statusinformation einer andern Zelle (PAE Y) evaluieren kann und dementsprechend die Daten verarbeitet. Um den Unterschied zu bestehenden Parallelrechnersystemen zu verdeutlichen, ist in Fig. 8a der Stand der Technik angegeben. Dabei ist ein Multiprozessorsystem gezeigt, dessen Prozessoren über einen gemeinsamen Datenbus (0803) miteinander verbunden sind. Ein explizites Bussystem für den synchronen Austausch von Daten und Status existiert nicht.

Abschließend soll angemerkt werden, daß je nach Aufgabe sowohl der Datenflußgraph, als auch der Kontrollflußgraph entsprechend dem beschriebenen Verfahren behandelt werden kann.

Erweiterungen in der Hardware gegenüber ~~PACT02~~P196 51 075.9 und ~~PACT04~~P196 54 846.2

Durch ~~PACT02~~P196 51 075.9 und ~~PACT04~~P196 54 846.2 ist der Stand der Technik in Bezug auf die Konfigurationseigenschaften von Zellen (PAEs) definiert.
Dabei soll auf zwei Eigenschaften eingegangen werden:
1. Einer PAE (0903) ist gemäß ~~PACT02~~P196 51 075.9 ein Satz von Konfigurationsregistern (0904) zugeordnet, der eine Konfiguration beinhaltet (Fig. 9~~8~~a).
2. Eine Gruppe von PAEs (0902) kann gemäß ~~PACT04~~P196 54 846.2 auf einen Speicher zum Speichern oder Lesen von Daten zugreifen (Fig. 9~~8~~b)

Aufgabe ist es,
a) ein Verfahren zu schaffen, das das Umkonfigurieren von PAEs beschleunigt und zeitlich von der übergeordneten Ladeeinheit entkoppelt, und
b) das Verfahren so auszulegen, daß gleichzeitig die Möglichkeit geschaffen wird über mehrere Konfigurationen zu sequenzen, und
c) gleichzeitig mehrere Konfgurationen in einer PAE zu halten, von denen immer eine aktiviert ist und zwischen verschiedenen Konfigurationen schnell gewechselt werden kann.

Entkopplung der Konfigurationsregister

Das Konfigurationsregister wird von der übergeordneten Ladeeinheit (CT) entkoppelt (Fig. 109), indem ein Satz von mehreren Konfigurationsregistern (10~~0~~901) verwendet wird. Immer genau eines der Konfigurationsregister bestimmt selektiv die Funktion der PAE. Die Auswahl des aktiven Registers wird über einen Multiplexer (~~0~~91002) durchgeführt. In jedes der Konfigurationsregister kann die CT beliebig schreiben, sofern dieses nicht die aktuelle Konfiguration der PAE bestimmt, d.h.

aktiv ist. Das Schreiben auf das aktive Register ist prinzipiell möglich, dazu stehen die in PACT10 beschriebenen Verfahren zur Verfügung.
Welches Konfigurationsregister von ~~09~~1002 selektiert wird kann durch verschiedene Quellen bestimmt werden:
1. Ein beliebiges Status-Signal oder eine Gruppe beliebiger Status-Signale, die über ein Bussystem (0802) an ~~09~~1002 geführt werden (Fig. ~~9~~10a). Die Status-Signale werden dabei von beliebigen PAEs generiert oder durch externe Anschlüsse des Bausteins zur Verfügung gestellt (vgl. Fig. 8).
2. Das Status-Signal der PAE, die von ~~09~~1001/~~09~~1002 konfiguriert wird, dient zur Selektion (Fig. ~~9~~10b).
3. Ein von der übergeordneten CT generiertes Signal dient zur Selektion (Fig. ~~9~~10c).
Dabei ist es ~~M~~möglich, wahlweise die eingehenden Signale (~~09~~1003~~, 0904, 0905~~) mittels eines Registers für einen bestimmten Zeitraum zu speichern und alternativ und wahlweise abzurufen.

Durch den Einsatz mehrere Register wird die CT zeitlich entkoppelt. Das bedeutet, die CT kann mehrere Konfigurationen "vorladen", ohne daß eine direkte zeitliche Abhängigkeit besteht.
Lediglich, wenn das selektierte/aktivierte Register in ~~09~~1001 noch nicht geladen ist, wird mit der Konfiguration der PAE so lange gewartet, bis die CT das Register geladen hat. Um festzustellen, ob ein Register eine gültige Information besitzt kann beispielsweise ein "Valid-Bit" (~~0906~~1004) pro Register eingeführt werden, das von der CT gesetzt wird. Ist 0906 bei einem selektierten Register nicht gesetzt, wird über ein Signal die CT zum schnellstmöglichen Setzen des Registers aufgefordert.

Das in Fig. ~~9~~10 beschriebene Verfahren ist einfach zu einem Sequenzer erweiterbar (Fig. ~~10~~11). Dazu wird ein ~~Mikrokontroller~~Sequenzer mit Instruktionsdekoder (~~10~~1101) zur Ansteuerung der Selektionssignale des Multiplexers (~~09~~1002) verwendet. Der Sequenzer bestimmt dabei abhängig von der aktuell selektierten Konfiguration (~~10~~1102) und einer zusätzlichen Statusinformation (~~10~~1103/~~10~~1104) die nächste zu selektierende Konfiguration. Dabei kann die Statusinformation
(a) der Status der Status-Signal der PAE, die von ~~09~~1001/~~09~~1002 konfiguriert wird sein (Fig. ~~10~~11a)
(b) ein beliebiges über 0802 zugeführtes Statussignal sein (Fig. ~~10~~11b)
(c) eine Kombination aus (a) und (b) sein.

~~Zum einfachen Verständnis kann 0901~~1001 kann auch als ~~ein~~ Speicher ~~betrachtet werden~~ausgestaltet sein, wobei ~~über~~ anstatt ~~09~~1002 ein Befehl von ~~10~~1101 adressiert wird. Die Adressierung ist dabei abhängig vom Befehl selbst und von einem Statusregister. Insoweit entspricht der Aufbau einer "von Neumann" Maschine, mit dem Unterscheid,
(a) der universellen Einsetzbarkeit, also den Sequenzer nicht zu verwenden (vgl. Fig. ~~9~~10)
(b) daß das Statussignal nicht von dem dem Sequenzer zugeordneten Rechenwerk (PAE) generiert werden muß, sondern von einem beleibigen anderen Rechenwerk stammen kann (vgl. Fig. ~~10~~11b).

Wichtig ist, daß der Sequenzer dabei Sprünge, insbesondere auch bedingte Sprünge, innerhalb von ~~09~~1001 ausführen kann.

Ein weiteres zusätzliches oder alternatives Verfahren (Fig. 12~~11~~) zum Aufbau von Sequenzern innerhalb ~~der genannten Bausteine~~von VPUs ist die Verwendung der internen

09.01.00

Datenspeicher (~~11~~1201, 0901) zum Speichern der Konfigurationsinformation für eine PAE oder eine Gruppe von PAEs. Dabei wird der Datenausgang eines Speichers auf einen Konfigurationseingang einer PAE oder mehrer PAEs geschaltet (~~11~~1202). Die Adresse (12~~11~~03) für 12~~11~~01 kann dabei von derselben PAE/denselben PAEs oder einer oder mehreren beliebigen anderen generiert werden.
Bei diesem Verfahren ist der Sequenzer nicht fest implementiert, sondern wird durch eine PAE oder eine Gruppe von PAEs nachgebildet.

09.01.00

Patentansprüche

Verfahren zum Ausführen von Programmen auf einem Baustein mit ein- oder mehrdimensionaler Zellstruktur, dadurch gekennzeichet,

daß Datenfluß- oder Kontrollflußgraphen in zeitlich getrennte Teilgraphen partitioniert werden und sequentiell auf den Baustein abgebildet und ausgeführt werden.

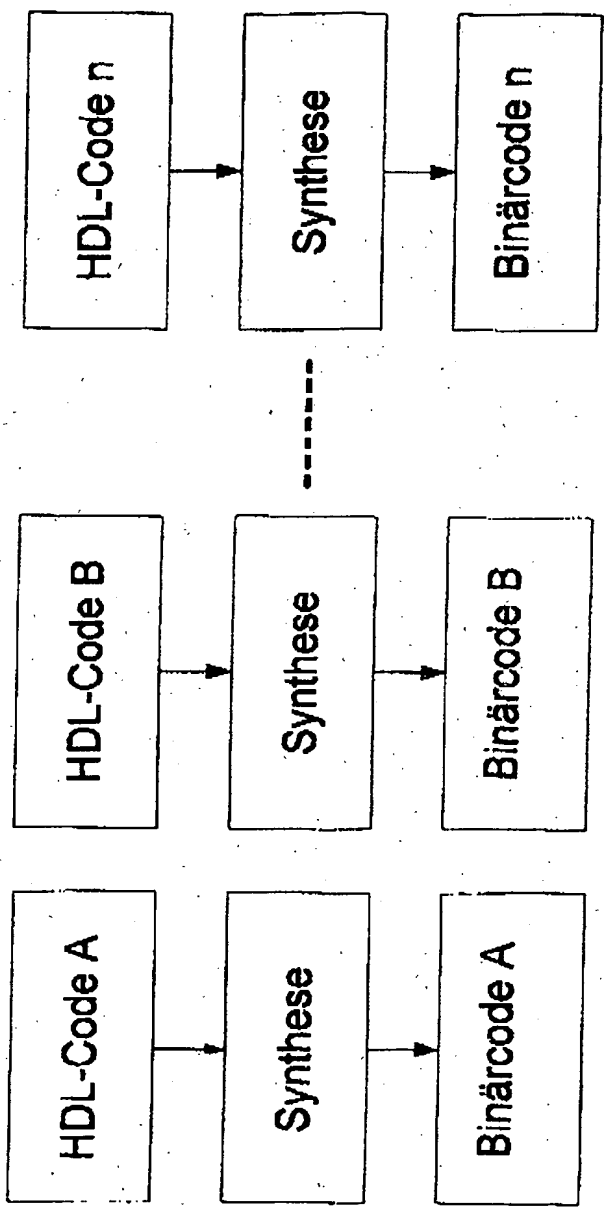


Fig. 1 Stand der Technik

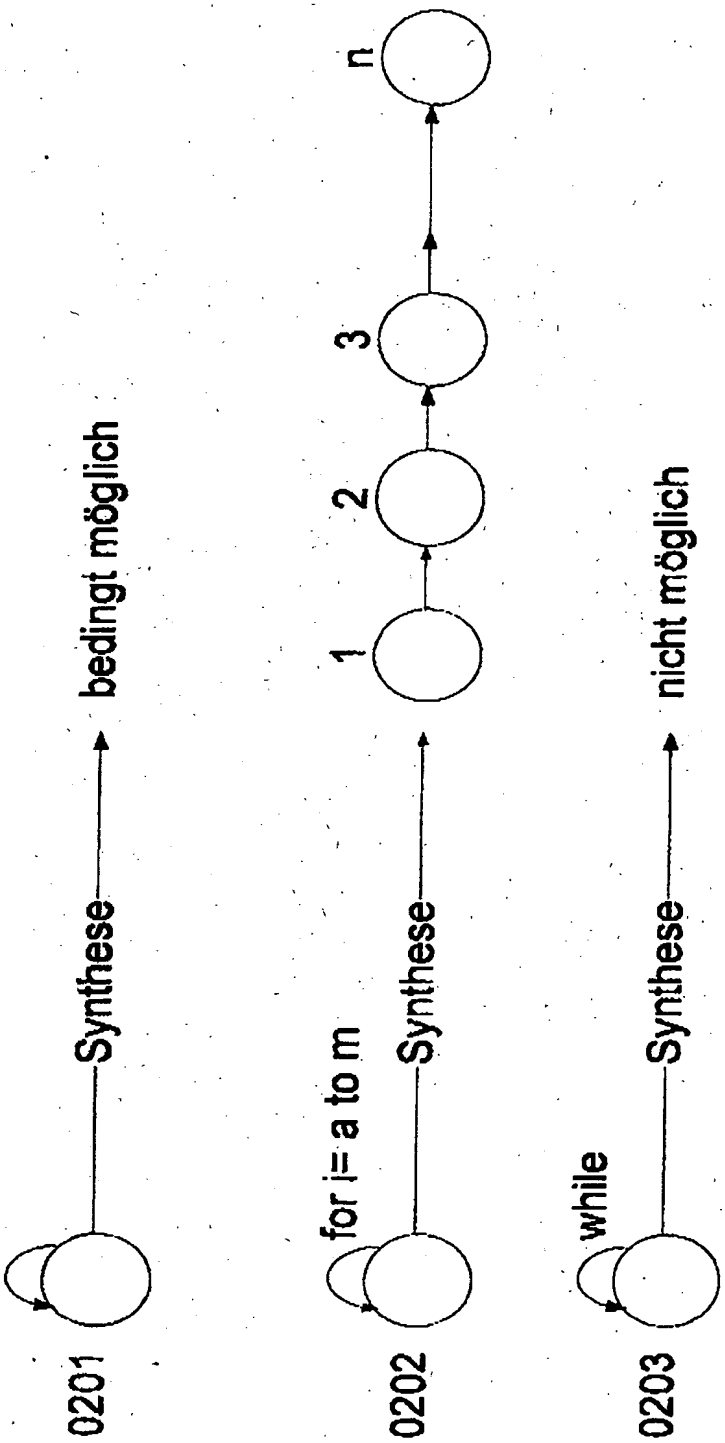


Fig. 2 Stand der Technik

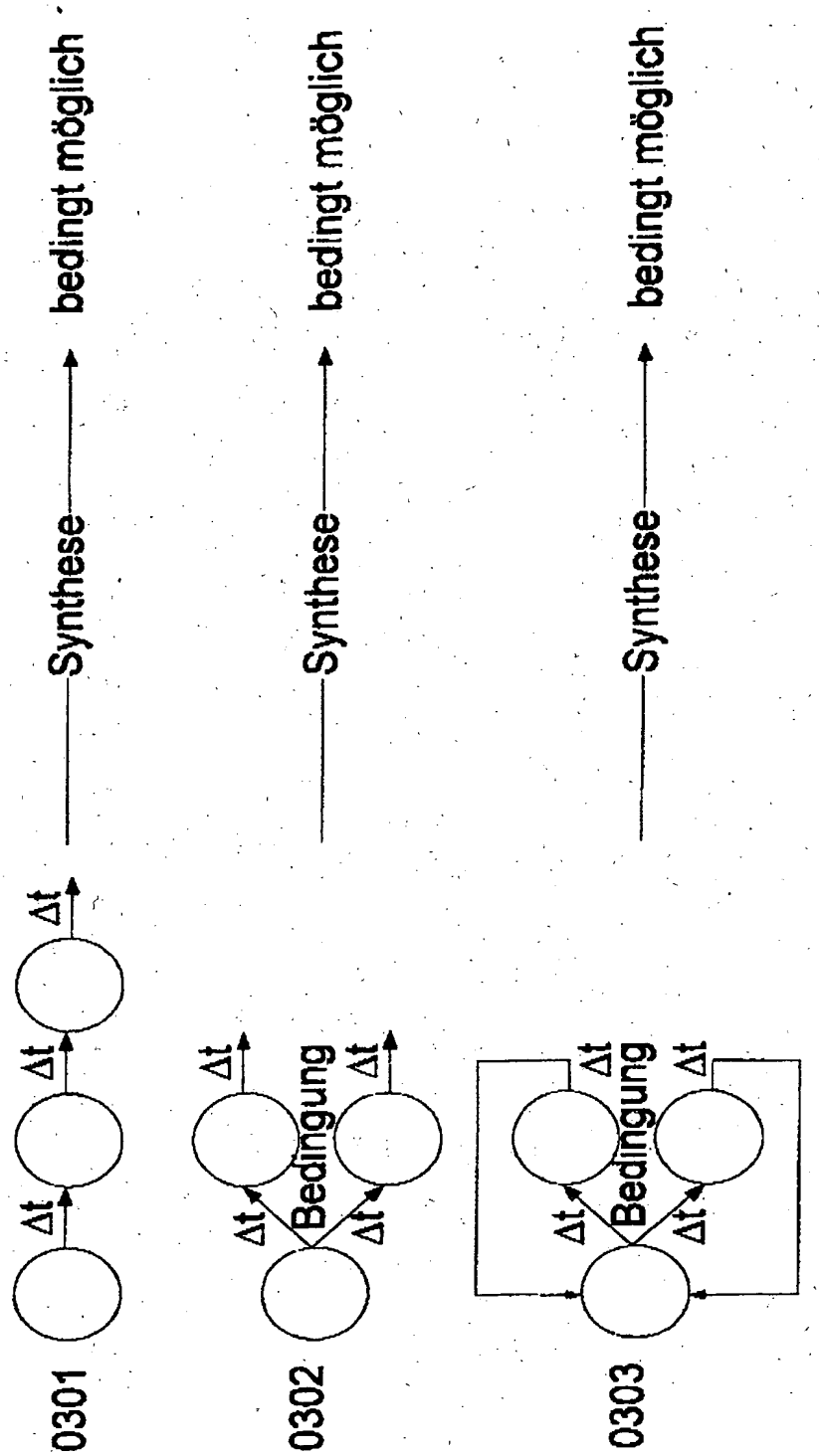


Fig. 3 Stand der Technik

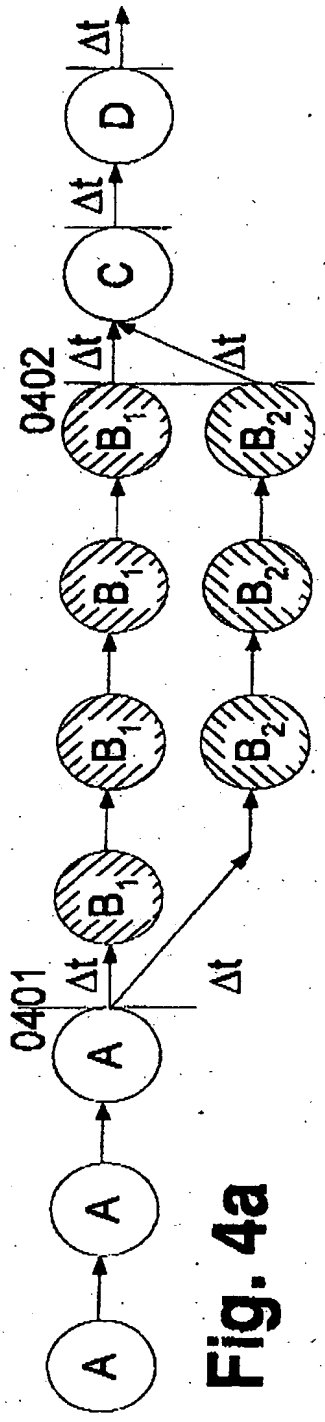


Fig. 4a

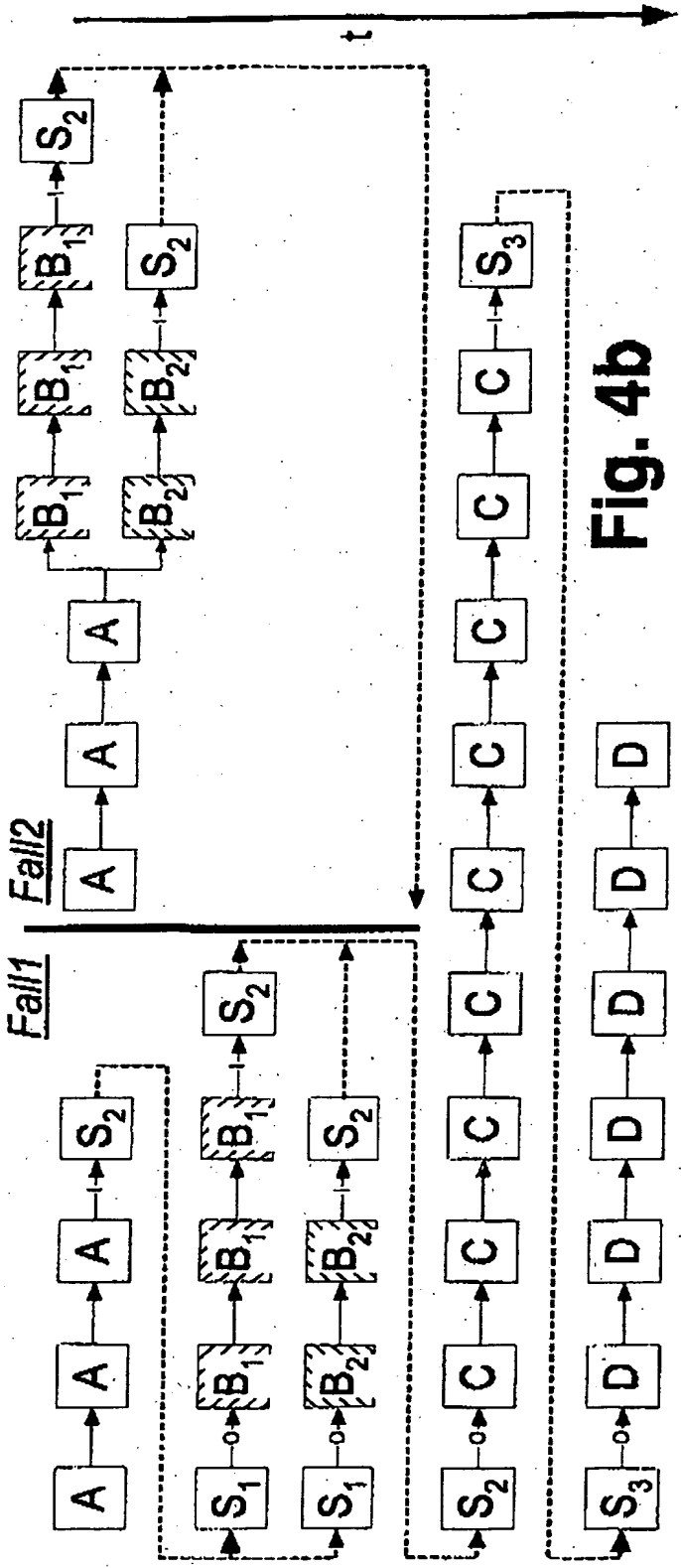


Fig. 4b

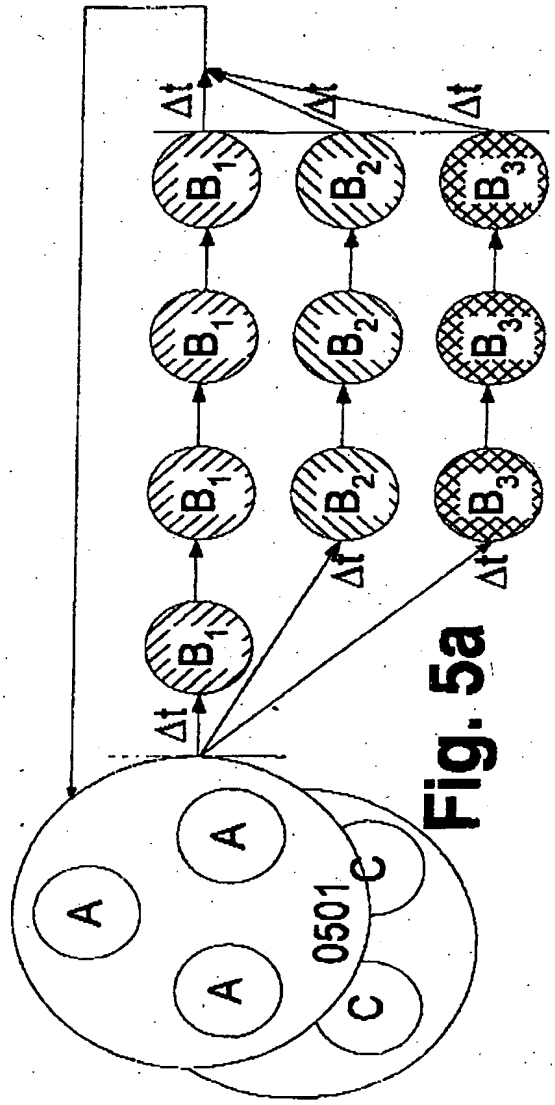


Fig. 5a

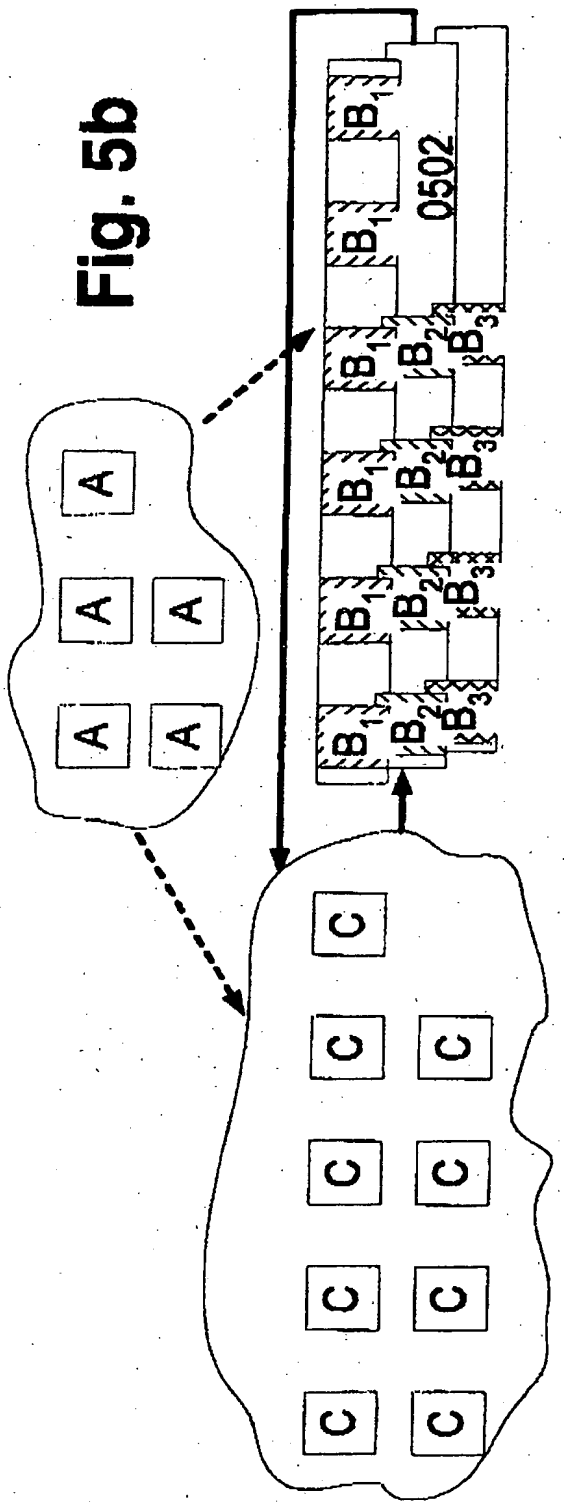


Fig. 5b

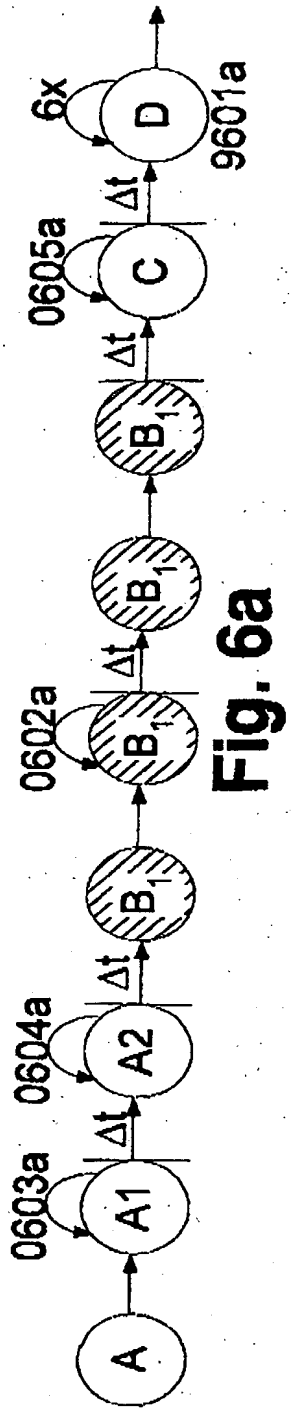

A
A1
A2
B_1
C
D
Δt
6x
0603a
0604a
0602a
0605a
9601a

Fig. 6a

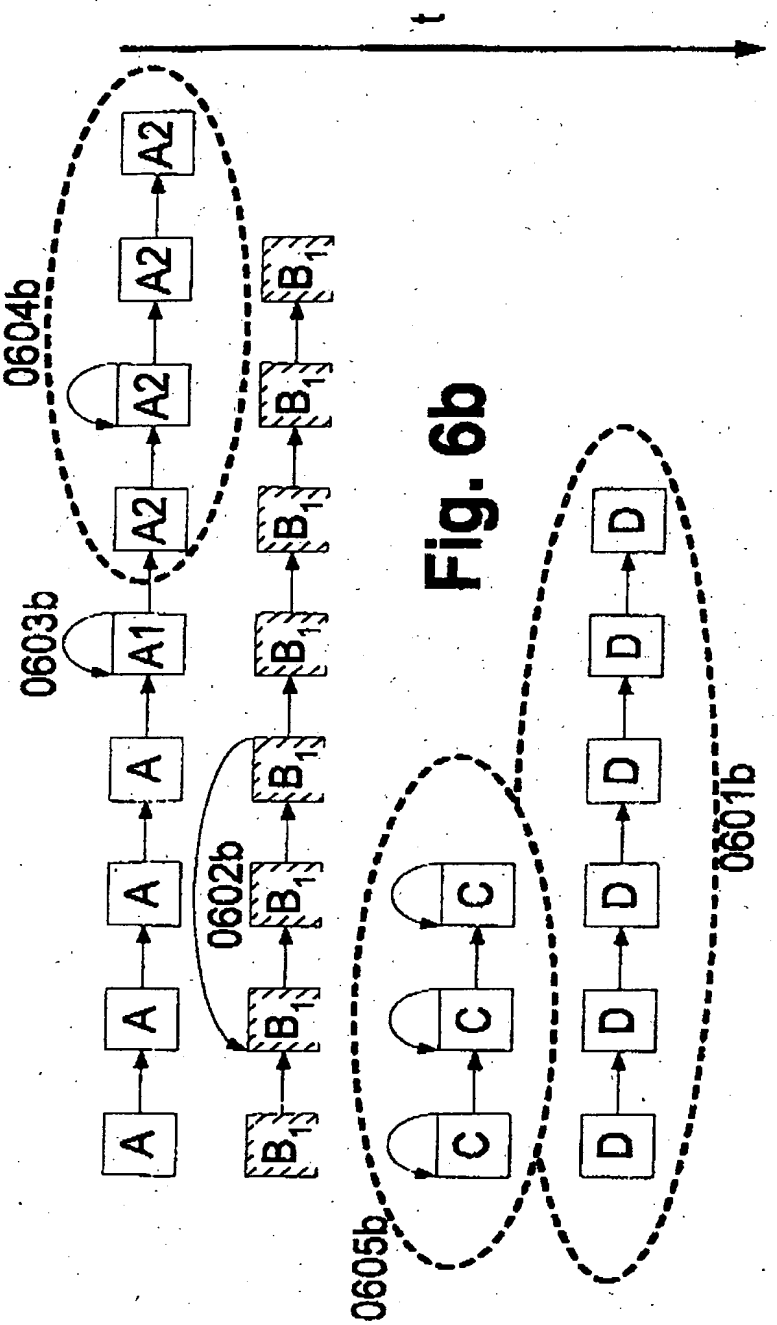

t
A
A1
A2
B_1
C
D
0603b
0604b
0602b
0605b
9601b

Fig. 6b

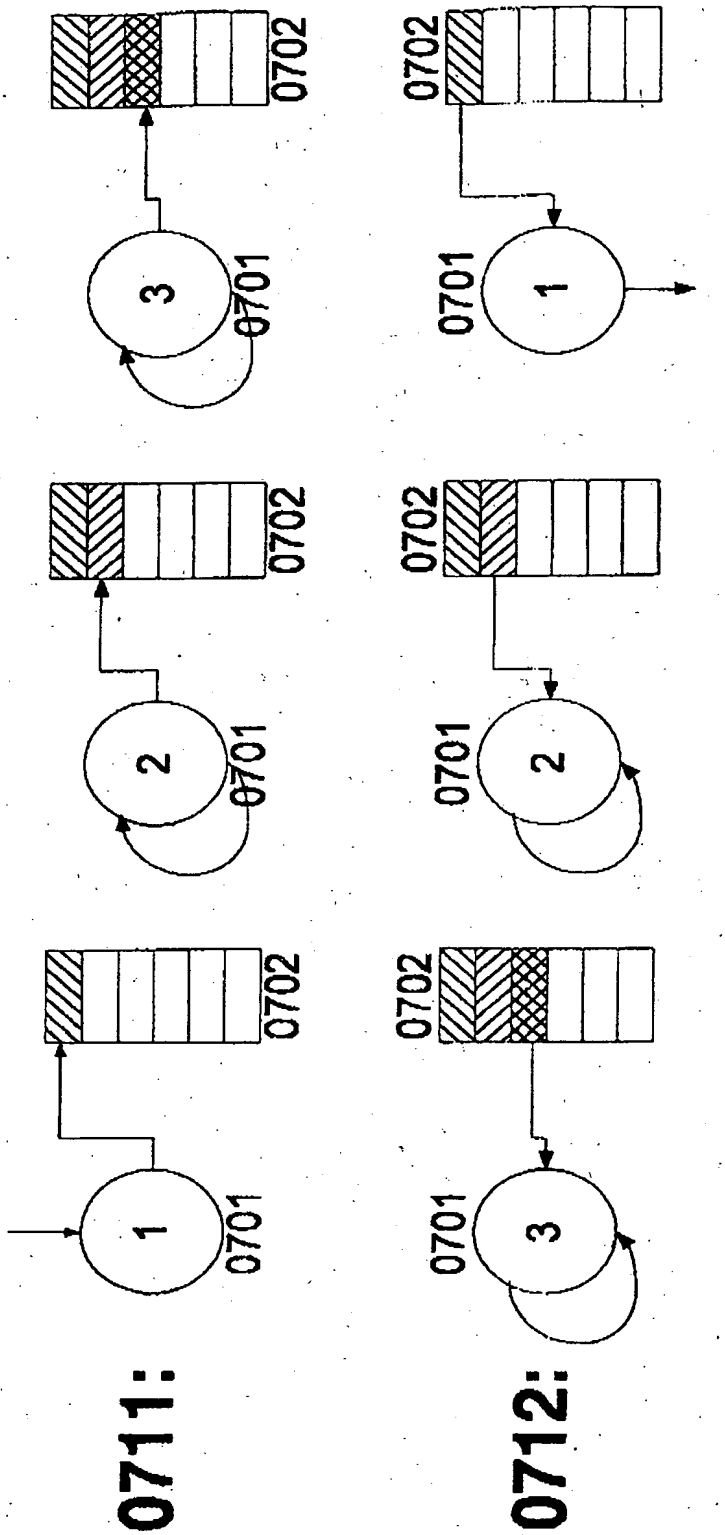


Fig. 7

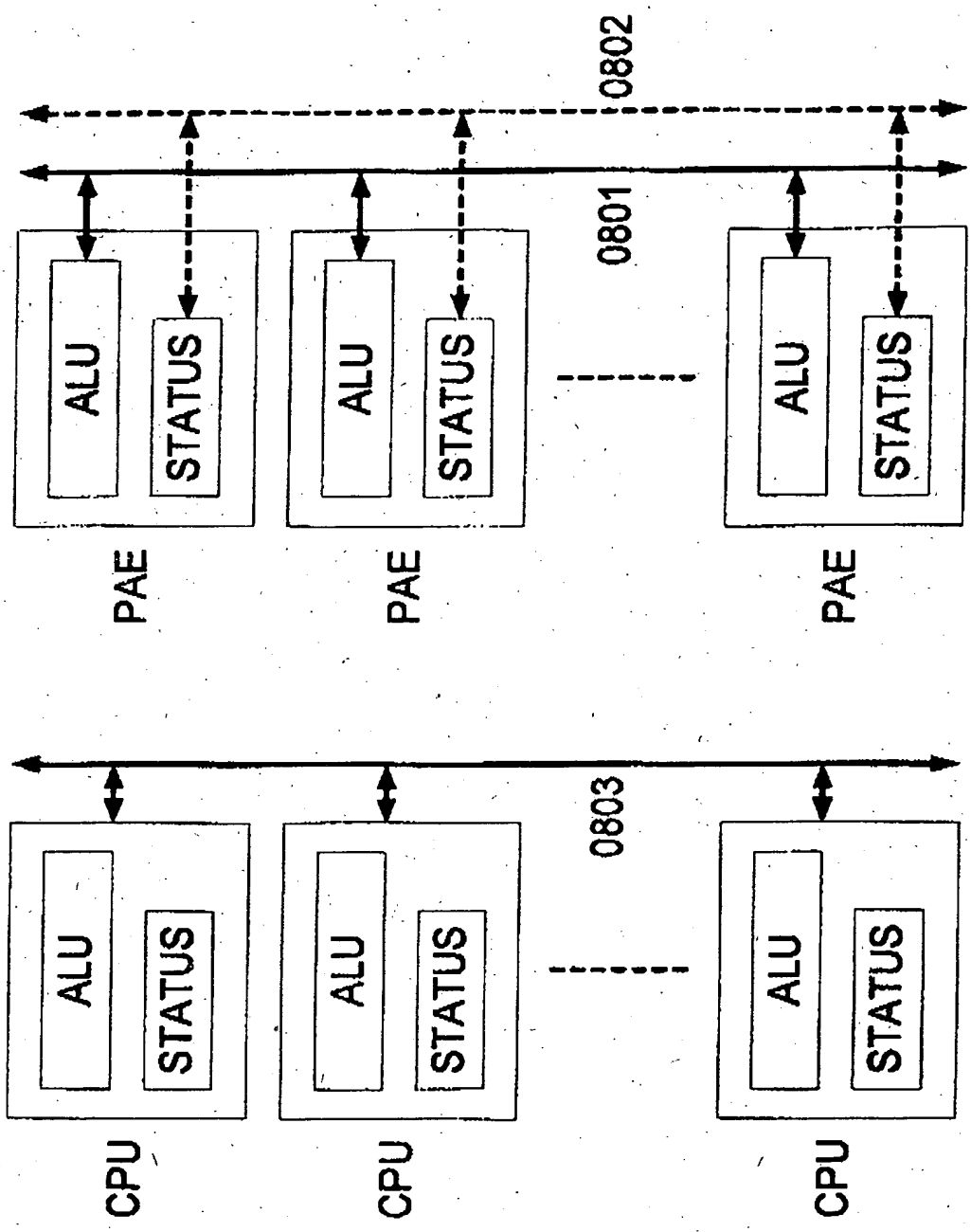


Fig. 8b

Fig. 8a Stand der Technik

Stand der Technik

Fig. 9a

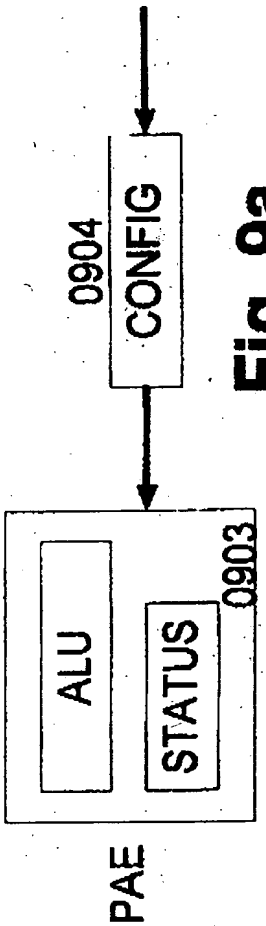


Fig. 9b

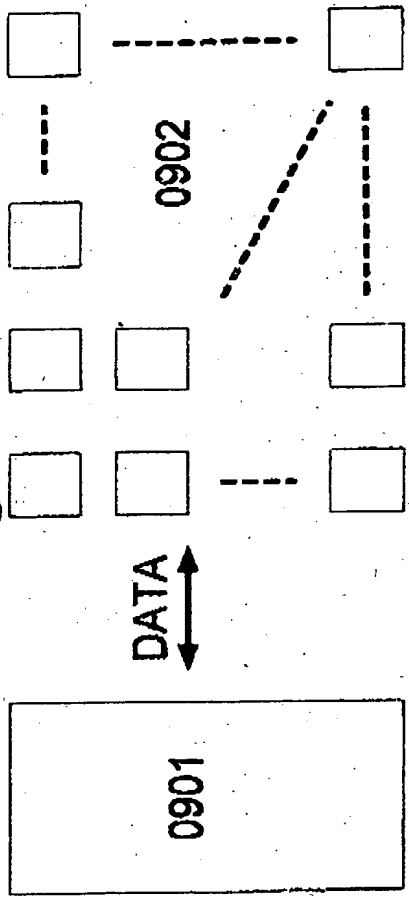


Fig. 10a

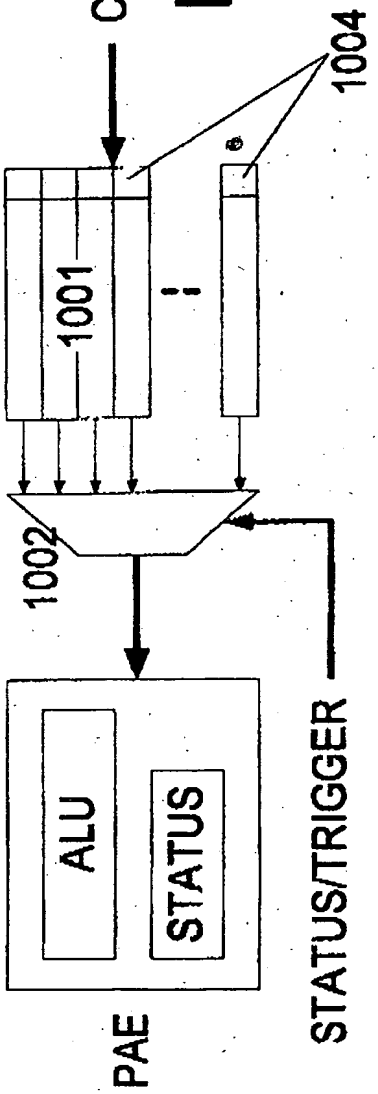


Fig. 10b

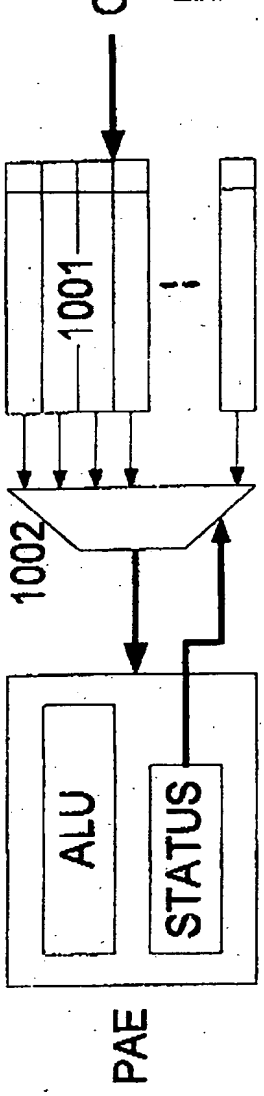


Fig. 10c

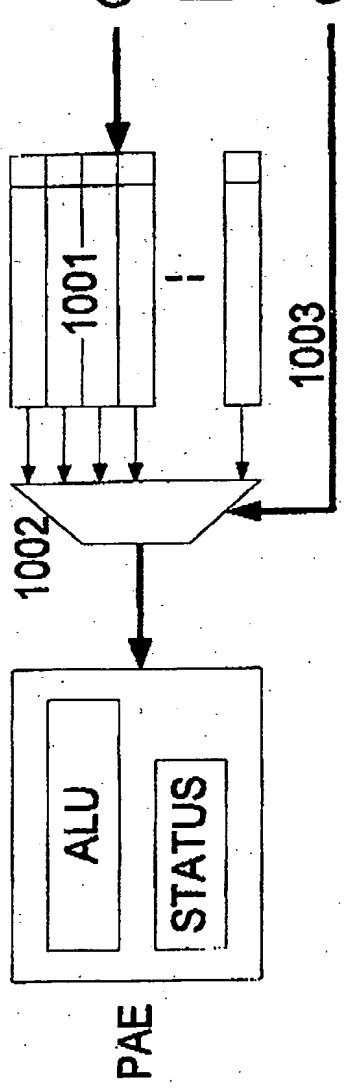


Fig. 11a

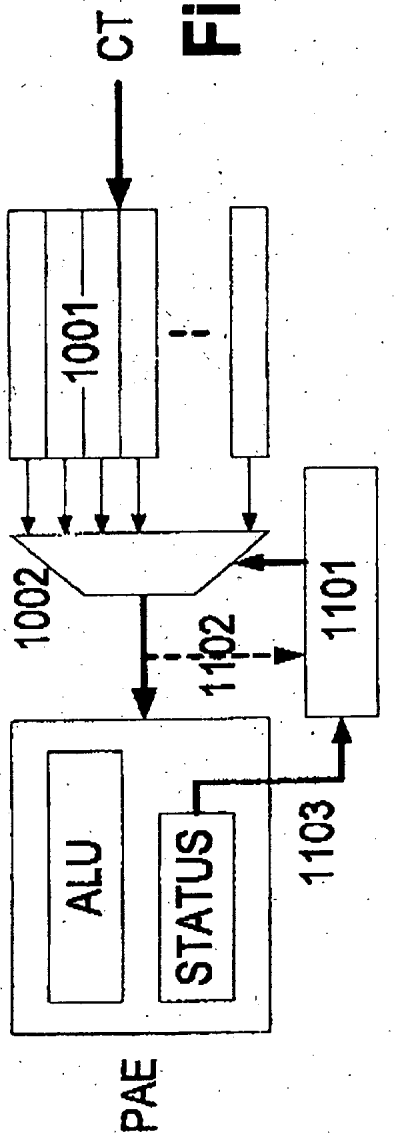
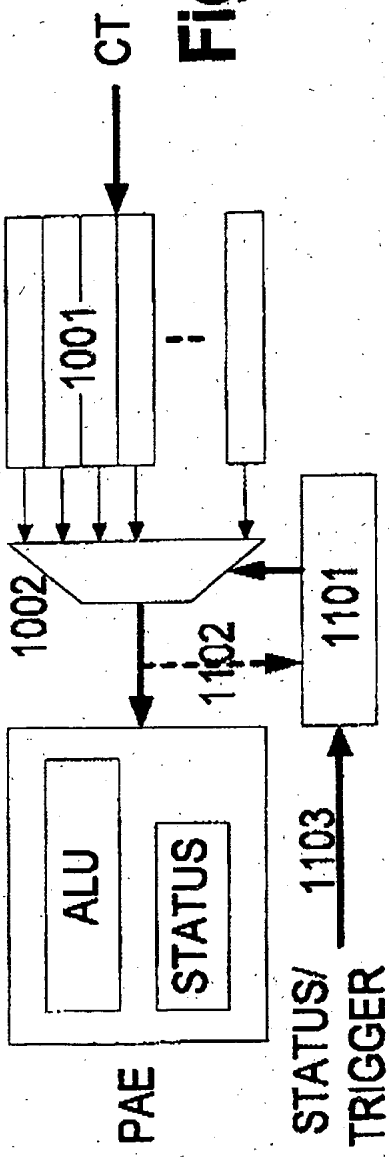


Fig. 11b



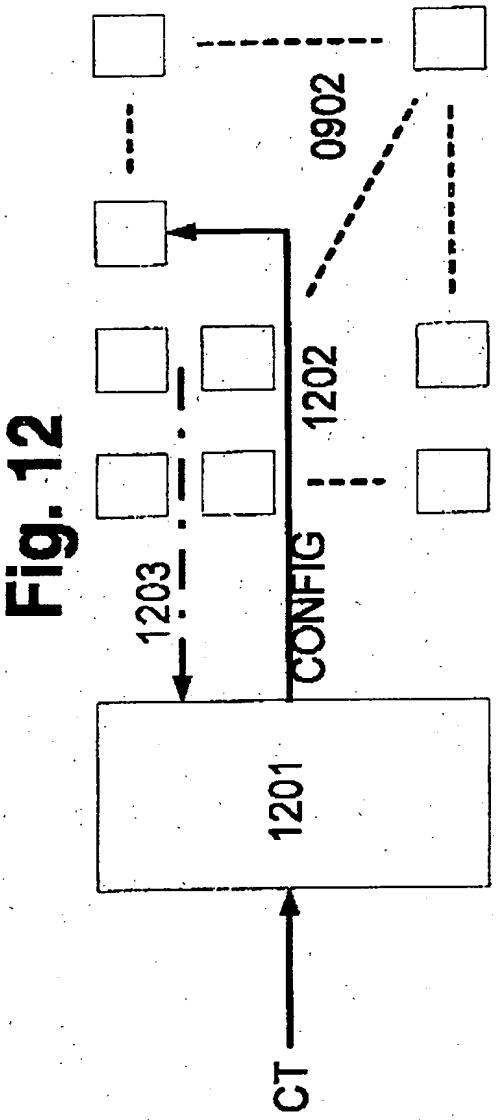


Fig. 12

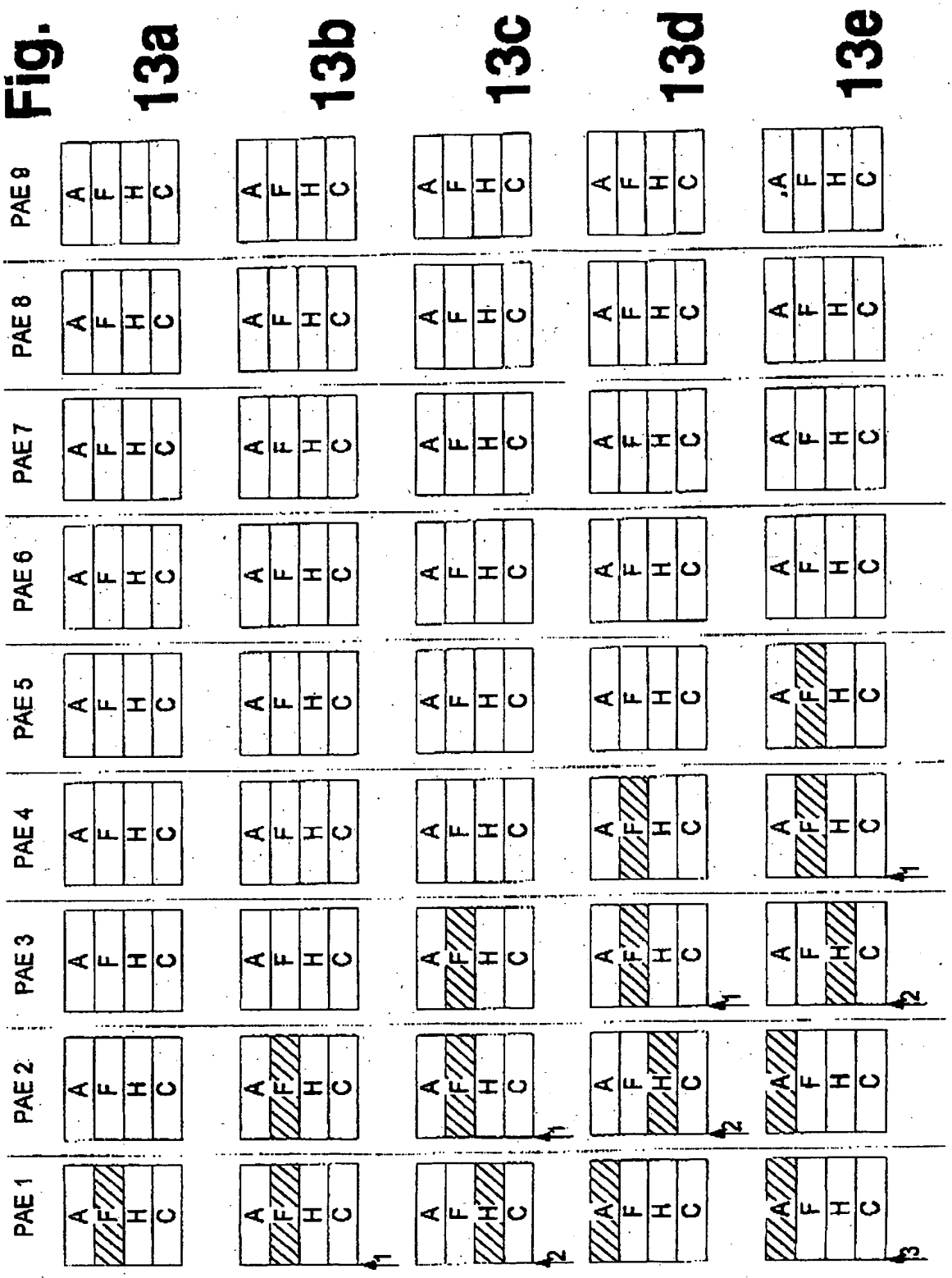


Fig.
13a
13b
13c
13d
13e
PAE 8
PAE 7
PAE 6
PAE 5
PAE 4
PAE 3
PAE 2
PAE 1
A
F
H
C

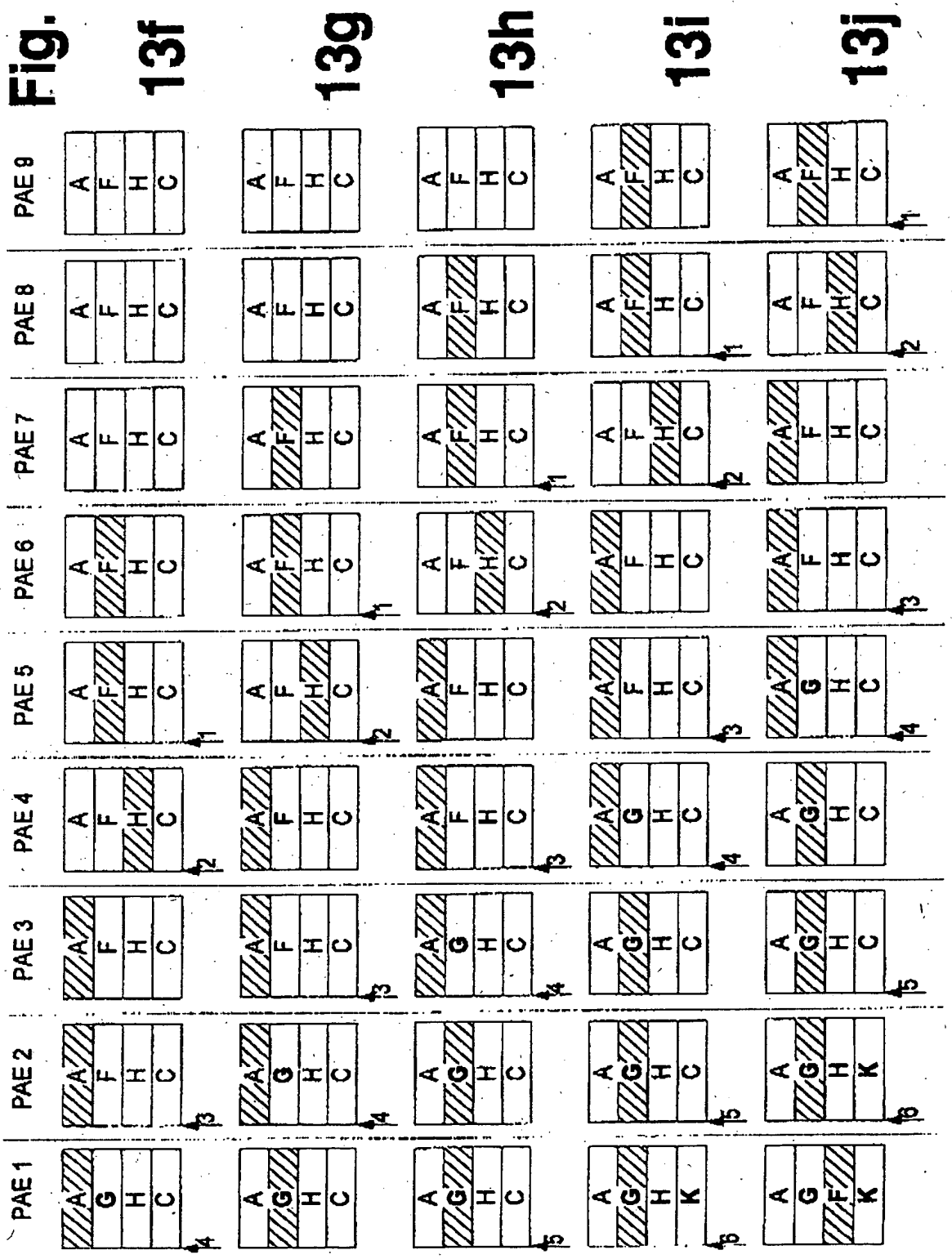
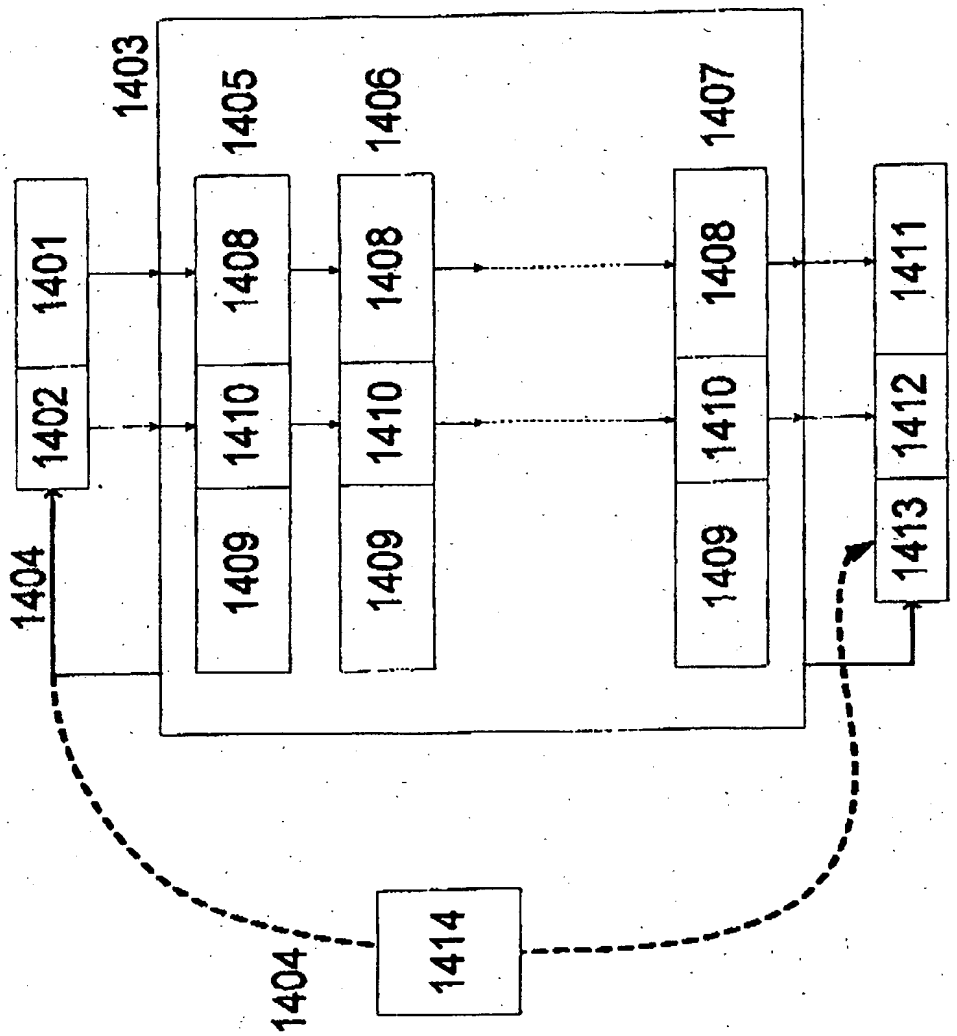


Fig.
13f
13g
13h
13i
13j
PAE 9
PAE 8
PAE 7
PAE 6
PAE 5
PAE 4
PAE 3
PAE 2
PAE 1

Fig. 14


1403
1405
1406
1407
1401
1402
1408
1410
1409
1408
1410
1409
1408
1410
1409
1411
1412
1413
1404
1414
1404

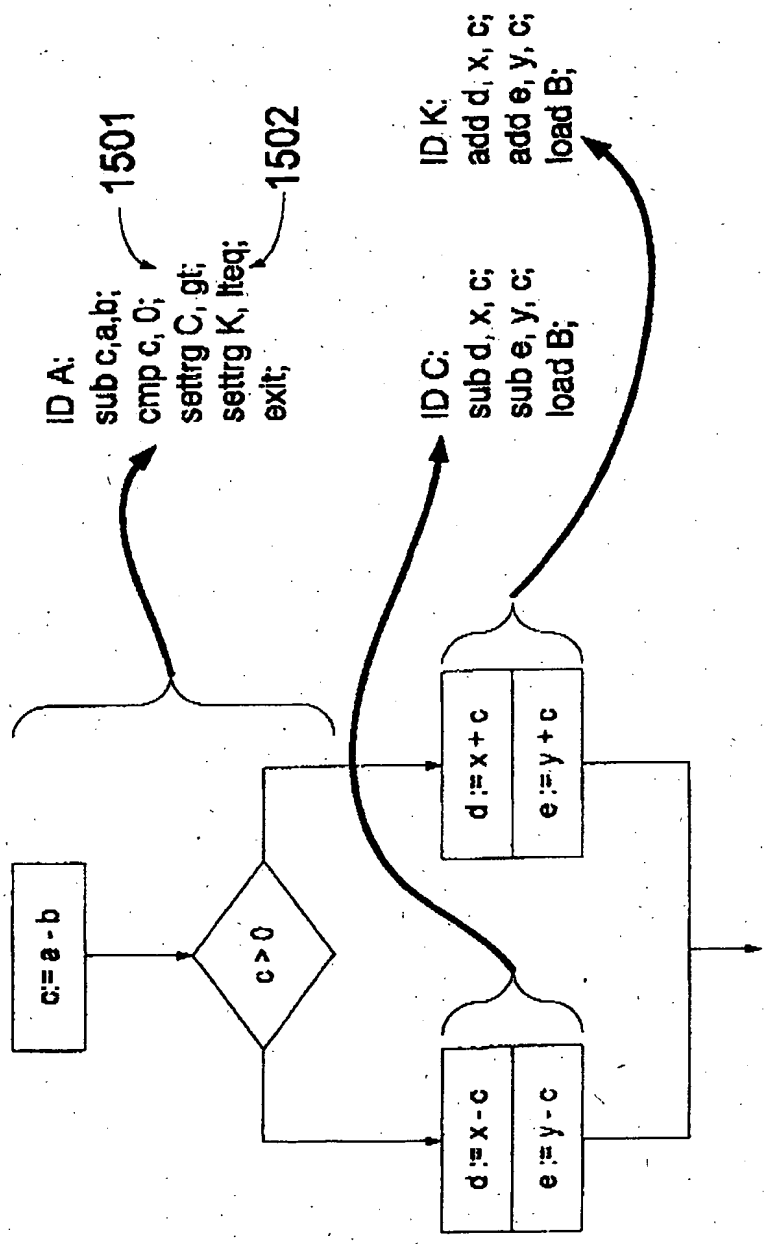

ID A:
sub c,a,b;
cmp c, 0;
settrg C, gt;
settrg K, lteq;
exit;
1501
1502
ID C:
sub d, x, c;
sub e, y, c;
load B;
ID K:
add d, x, c;
add e, y, c;
load B;
c:= a - b
c > 0
d := x + c
e := y + c
d := x - c
e := y - c

Fig. 15

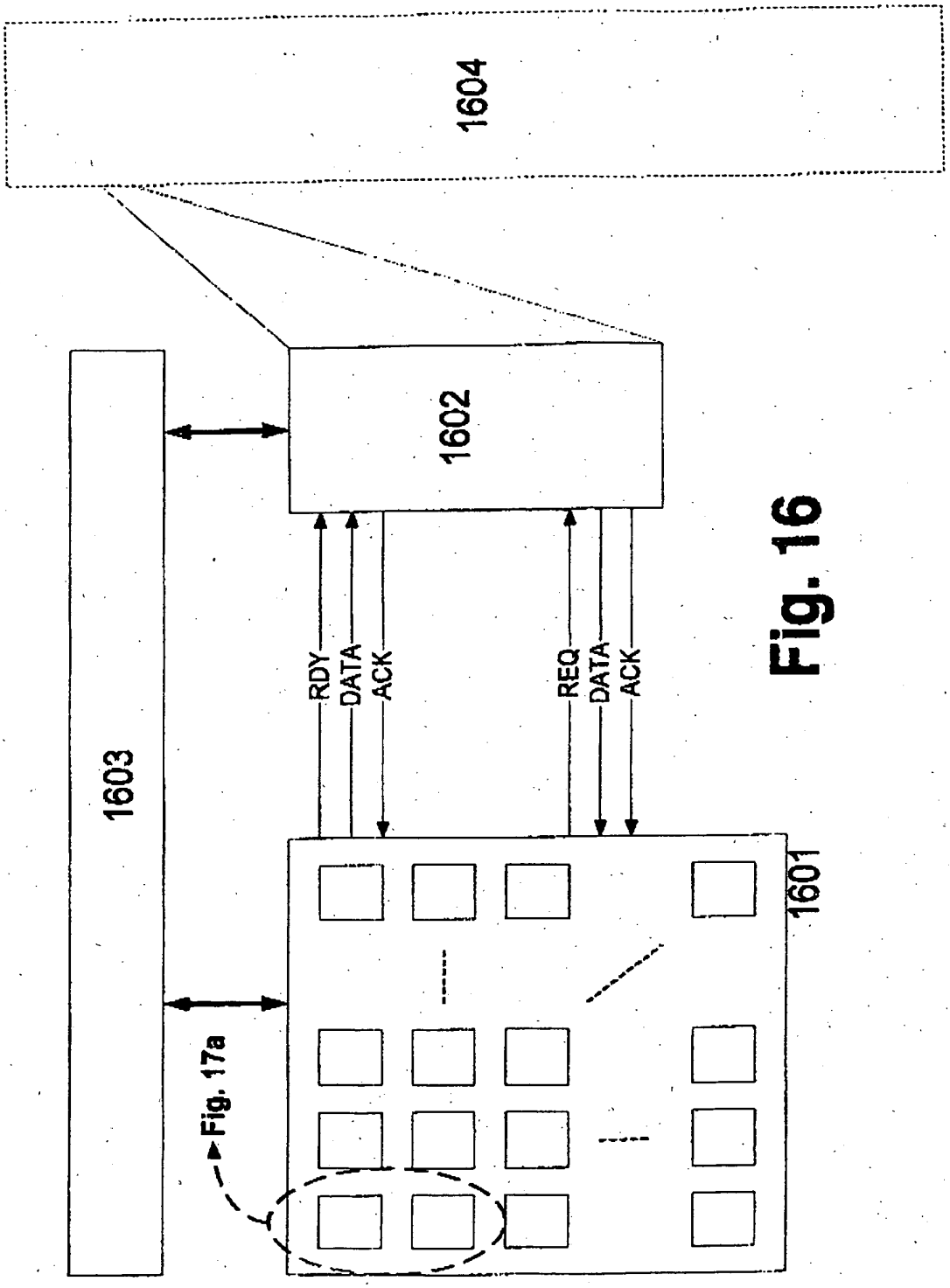


Fig. 16

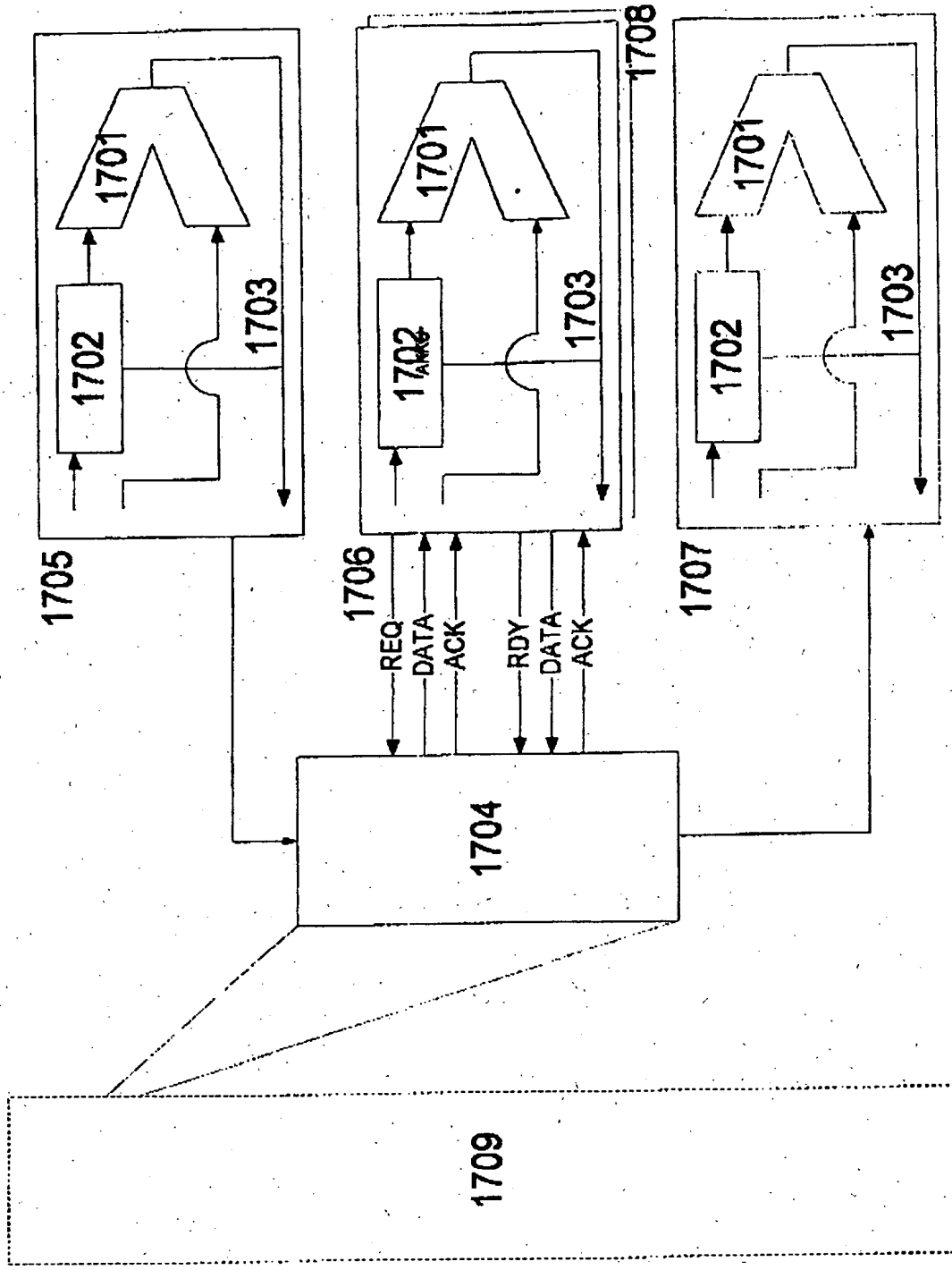


Fig. 17a

Fig. 17

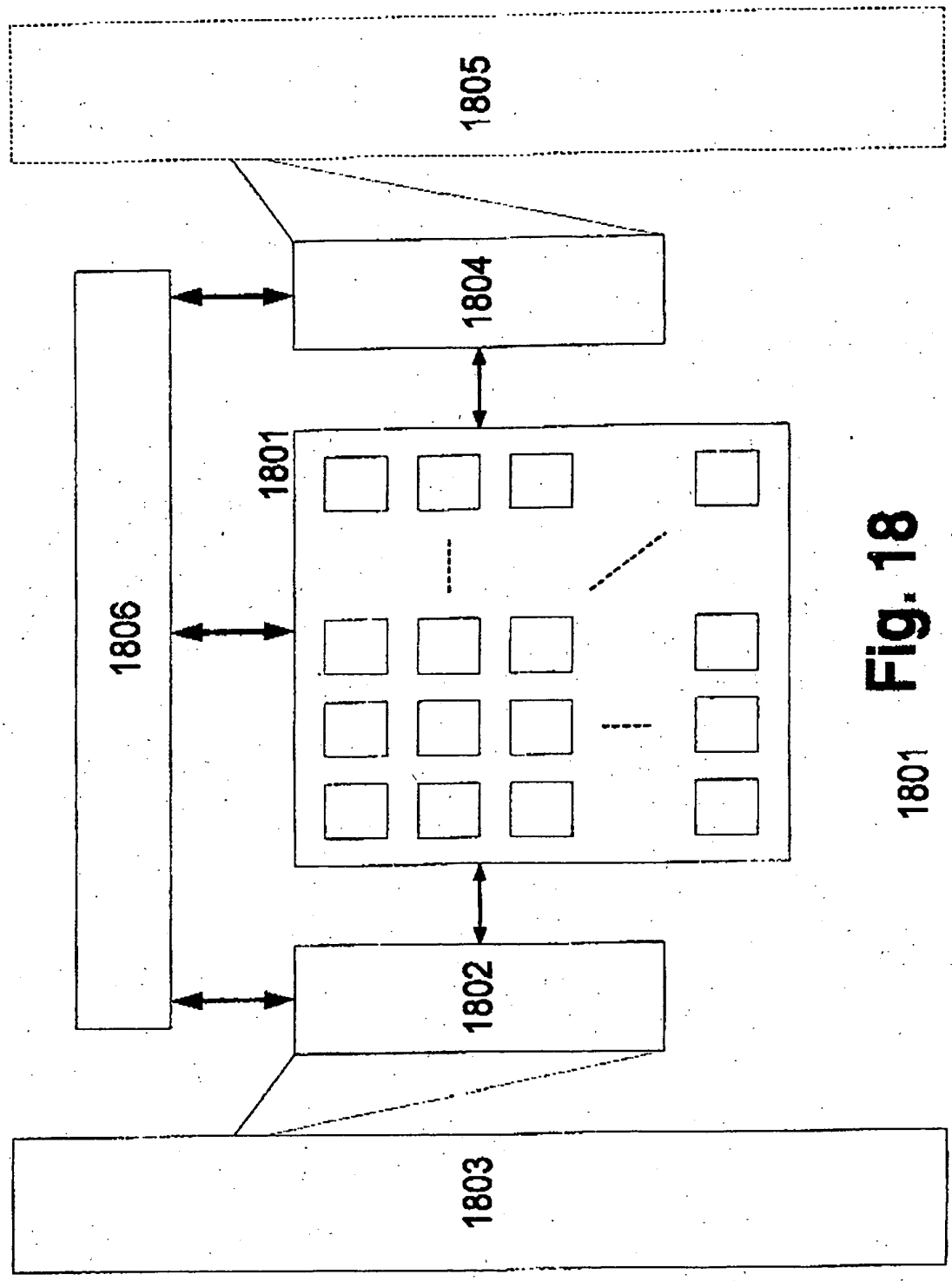


Fig. 18

1801

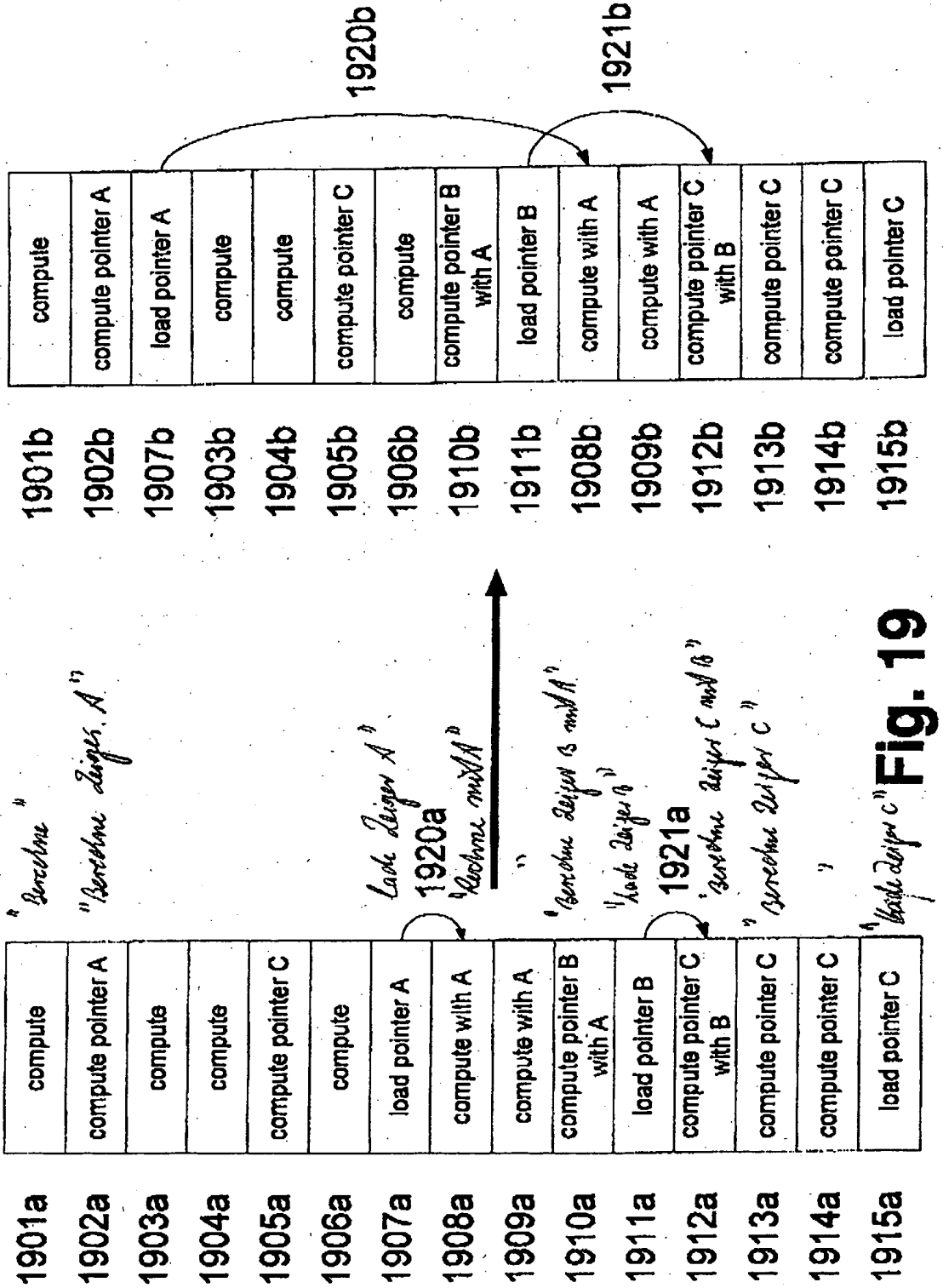


Fig. 19

GESAMT SEITEN 54